

HP OpenView Service Delivery Designer

Reference Guide

Software Version: 2.2

Microsoft Windows Server 2003, Windows XP



Manufacturing Part Number: T3288-90000
March 2004

© Copyright 2004 Hewlett-Packard Development Company, L.P.

Legal Notices

Warranty

Hewlett-Packard makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices

© Copyright 2004 Hewlett-Packard Development Company, L.P.

No part of this document may be copied, reproduced, or translated into another language without the prior written consent of Hewlett-Packard Company. The information contained in this material is subject to change without notice.

Trademark Notices

OpenView® is a trademark of Hewlett-Packard Development Company, L.P.

Microsoft®, Windows NT®, Windows® 2000, Windows® XP, Windows Server™ 2003, Windows®, SQL Server™ 2000, and Visio 2002, are U.S. registered trademarks of Microsoft Corporation.

WinZip® is a registered trademark of WinZip Computing, Inc.

Support

Please visit the HP OpenView web site at:

<http://openview.hp.com/>

There you will find contact information and details about the products, services, and support that HP OpenView offers.

You can go directly to the HP OpenView support web site at:

<http://support.openview.hp.com/>

The support site includes:

- Downloadable documentation
- Troubleshooting information
- Patches and updates
- Problem reporting
- Training information
- Support program information

Contents	5
The SDD Guide	11
Introduction	11
Audience	11
Prerequisites	11
Chapters Summary	11
Related Documents	12
SDD Overview	13
Packages and Solutions	13
Jobs	13
Execution Units	14
The Package Designer	14
EU and Package Versions	14
Editing a Package	14
Solution Rollout	15
GUI & CLI Customization	15
Additional Tools	15
SDD Basic Concepts	17
Service Delivery Controller Overview	17
Service Delivery Building Blocks	18
Solutions	18
Solution Components	18
Packages	18
Package Anatomy	18
The XML File	19
The C# File	19

The Message File	19
Working with Resources.....	19
Binding to a Resource	20
Managing Dependencies.....	20
Execution Units (EUs)	20
EU Components	21
EU Registration.....	22
Capturing State Information.....	22
EU Creation Flow.....	22
Execution Unit Examples.....	25
Sample Execution Unit One: SampleEU	25
SampleEU Class Elements	25
Sample Execution Unit Two: AddShare	30
AddShare EU Class Elements	30
SDD Tools	35
The Package Designer	35
Overview of the Package Designer	35
Launching	36
Opening a Package.....	37
Saving a Package	39
Saving a Package Locally	39
Editing a Package	40
Edit Log Messages.....	40
Import Message File	41
Preview.....	41
Updating a Package to Latest EU Revisions	42
Package Designer Shapes.....	43
Versioning Overview.....	44
Introduction to Versioning.....	44
Revisions are Persistent.....	44
Package Default Revision	45
EU Default Revision	45
Propagating an EU Update to a Package	46
A Versioning Example	51

Versioning in the Service Delivery Controller Console 51

Solution Rollout 53

 Requirements form Solution Rollout 53

 Introduction to Solution Rollout 53

 Solution Rollout Overview 53

 Solution Rollout Details 53

 Solution Rollout and Versioning 56

Additional Command Line Tools..... 57

 EU Test Tool..... 57

 Short Description 57

 Description 57

 Location and Command..... 57

 Security Considerations 57

 Usage..... 57

 Parameters 57

 Example 58

 Resource Dictionary Tool 58

 Short Description 58

 Description 58

 Location and Command..... 58

 Security Considerations 59

 Usage..... 59

 Parameters 60

 Repository Tool 60

 Short Description 60

 Description 60

 Location and Command..... 60

 Security Considerations 60

 Usage..... 60

 Example 60

SDD—UI Job Options Modifications 61

 UI Job Options Overview..... 61

 Fixed vs Dynamic Input Parameters 62

 GUI Job Options Modification Example—Change Default Setting 62

 GUI Job Options Modification Example—Add New Input Parameter 65

 CLI Job Options Modifications..... 70

SDD Security	73
Package Designer Security	73
Solution Rollout Security	73
EU Registration Security	74
SDD Tools Security Considerations	74
SDD Execution Units	75
ADS EUs.....	75
ADSDeployValidator.....	75
CopyFile	75
CopySequence	76
CopySysprepFiles	76
ExtractMachineName	76
GenerateCaptureSequence	76
GenerateDeploySequence	76
GenerateSysprepSequence	76
GetAdminMAC	76
GetADSInstallDir	76
GetBootPartition	77
GetDeviceName	77
GetDeviceNameByServerName.....	77
GetDiskLayout.....	77
InstallAgentValidator	77
PrepareImgVariables.....	77
RegisterCSRImage	77
RegisterDevice	77
RunRemoteCommand.....	77
RunSequenceValidator	78
SetDefaultTemplate.....	78
SetDeviceVariables	78
SubmitJob.....	78
SysprepValidator	78
Domain EUs.....	78
ConsolidationCheck	78
FileServices EUs	78

AddShare.....	78
AddTempShare	79
AssertFailure	79
AssertSuccess.....	79
CopyFiles.....	79
DelShare.....	79
DfsAdd.....	79
DfsAddDomainRoot.....	79
DfsAddStandaloneRoot.....	79
DfsRemove.....	79
DfsRemoveDomainRoot.....	80
DfsRemoveStandaloneRoot.....	80
FileShareMoveValidate	80
ForceRemoveDir	80
GetNestedShares	80
GetShareInfo	80
MakeRemoteDir	80
NestedShareCheck	80
RemoveDir.....	80
SetFileAttributes	81
SetFileSecurity	81
SetShareInfo	81
WaitEU	81
Machine EUs	81
AddLocalGroups.....	81
ComputerNameAdd.....	81
ComputerNameDel.....	81
EnumLocalGroups.....	82
FileSharesConsolidateCheck.....	82
FixServerName	82
GenerateNewServerName	82
Rename	82
RenameNT4.....	82
Shutdown.....	83
ShutdownNT4.....	83
ValidateUser	83

Walkthrough Examples	85
Changing a Service Delivery Controller Solution: Overview	85
Walkthrough Example A: Changing an Existing Solution—Removing an EU.....	86
Identifying the Package to Be modified.....	86
Identifying Necessary Changes	86
Identifying EUs to Be Modified/Changed	87
Saving the Modified Package.....	87
Testing the New Package	88
Walkthrough Example B: Changing an Existing Solution—Adding a Prefix to a Moved Share....	88
Identifying the Package to Be Modified.....	88
Identifying Necessary Changes	88
Identifying the Precise Change Location.....	89
Saving the Modified Package.....	99
Testing the New Package	99
 Appendix A – SampleEU Code and Markup	 101
SampleEU Code	101
SampleEU Markup.....	105
 Appendix B – AddShare EU Code and Markup.....	 107
AddShare Code	107
AddShare Markup	110

The SDD Guide

Introduction

This guide provides information on the SDD set of tools that enable the customization of HP OpenView Service Delivery solutions.

Audience

This guide is intended for IT Architects and Administrators, IT professionals, Data Center Managers, and developers, creating, enhancing or modifying HP OpenView Service Delivery Controller solutions using the Service Delivery Designer.

Prerequisites

To take full advantage of the SDD you must be familiar with the HP OpenView Service Delivery Controller. Basic understanding of programming and operating system concepts is also assumed.

Chapters Summary

Chapter 1 – The SDD Guide, introduces the overall purpose and structure of the SDD Guide.

Chapter 2 – SDD Overview, provides a high-level overview of the HP OpenView Service Delivery Designer, its components and tools.

Chapter 3 – SDD Basic Concepts, describes the basic concepts involved in the SDD product.

Chapter 4 – Execution Units, provides a guide through the EU files, from the C# class definitions to the markup file definitions.

Chapter 5 – SDD Tools, walks you through using the Package Designer tool demonstrating how to capture and customize an IT organization’s workflow.

Chapter 6 – SDD Security, addresses the model to use to modify, save, and rollout packages and solutions.

Chapter 7 – SDD Execution Units, describes the Execution Units that are part of the library that ships with the HP OpenView Service Delivery Controller.

Chapter 8 – Walkthrough Examples, provides examples and information needed to modify existing Service Delivery Controller Solutions using the Package Designer.

Related Documents

Refer to this site for additional information: <http://support.openview.hp.com/>

SDD Overview

The HP OpenView Service Delivery Designer (Service Delivery Designer or SDD) is a development environment made up of a set of tools, samples, and documentation that allow for the customization of HP OpenView Service Delivery Controller solutions.

Typically, the SDD user is the IT architect or administrator familiar with their company's IT workflow process. The skills required to take full advantage of this release of the Solution Builder can be satisfied with a familiarity of the Microsoft .NET environment and the ability to program in one of the .NET languages with primary focus being on C#. However, the SDD is a powerful tool that can be used by any administrator familiar with workflows and possessing basic programming skills.

Packages and Solutions

A key component of the Service Delivery Controller is the "Package." An SDC Package provides the manifestations of the logic for a solution, or, in other words, its workflow. For example, the File Sharing Services Migrate Package contains the detailed steps required (workflow) to migrate a share from source to destination. Steps, or actions, are performed by execution units (EUs) and/or other sub-packages. Thus a package is nothing more than a collection of execution units (EUs) and/or sub-packages connected in a flowchart.

A Package or a collection of Packages put together in a workflow creates a solution. For example, the File Sharing Services Migrate solution is a workflow consisting of Packages.

Jobs

Jobs are instantiations of package executions on specific resource instances. For more details on Jobs see the HP OpenView Service Delivery Controller documentation.

Execution Units

Execution Units (EUs) provide Service Delivery Controller Packages with the modularity, isolation, security, and robustness necessary for them to function. EUs execute in their own context and any failures they may encounter or cause are limited to their scope. This is achieved by the SDC Intelligent Automation Engine (IAE), which ensures EUs cannot have unwanted side effects on the rest of the system.

Typically EUs encapsulate an atomic (and logical) step. For example, one EU is used to read the permissions off a (source) share. Another EU is used to apply them to a (destination) share. Another EU lists DFS links, and so forth. There is no precise definition on the scope of an EU, however HP has made extended effort to keep EUs as small and well-defined as possible.

EUs are not only responsible for executing an operation but also for committing their work in the case of an error-free execution, and for rolling back their work in the case of errors, either by them or by other EUs in the Package.

The Package Designer

The Package Designer is one of the key components of SDD and lies at the heart of the product. The Package Designer provides a simple tool to capture an IT organization's workflow and customize it. Workflows provided by HP can be extended, modified, customized, and personalized with existing EUs or with the use of custom EUs, to meet an IT organization's needs. These workflows can then be used, by any appropriate user in the organization, repeatedly and with predictable, repeatable, logged and verifiable results.

EU and Package Versions

HP OpenView Service Delivery supports EU and Package versioning (revisions). This allows the SDD users to select the solution revision they want to execute, where appropriate, and they can define a package to use a specific EU revision, or update the package to use the latest EU revisions available.

All EU and Package revisions are stored in the Service Delivery Controller Repository.

Editing a Package

Extending, enhancing or changing a Package to meet custom requirements is done by the Package Designer tool. This graphical tool allows the editing of a package in a visual and easy to use manner.

Solution Rollout

Another key component of the SDD is the Solution Rollout feature. In a typical solution creation process, a solution moves from the development environment to a test environment and finally to a production environment, with possible intermediate steps. The Solution Rollout feature supports this process.

GUI & CLI Customization

The SDD allows, with the use of the Package Designer, for simple customization of solution options as they appear in the graphical user interface (GUI) and the command line interface (CLI).

Additional Tools

The SDD provides, in addition to the Package Designer and Solution Rollout, a number of additional tools including the EU registration, EU test, and other tools as described in the Tools section.

SDD Basic Concepts

The SDD provides the user with the tools, documentation, and samples to create customized solutions for their administration needs with the HP OpenView Service Delivery Controller.

Service Delivery Controller Overview

HP OpenView Service Delivery Controller provides an information technology, resource lifecycle management platform that measurably increases the efficiency and reduces the risk and time required to manage today's complex IT infrastructures. For the modern, agile datacenter, Service Delivery Controller's solution improves the quality of server operations while it drives down the overall cost of IT operations.

Service Delivery Controller delivers innovative Lifecycle Management capabilities for that optimize an organization's server resource environment for a business advantage. Utilizing a unique Intelligent Automation Engine (IAE), the Service Delivery Controller software automates common workflows, incorporates best practices, and adapts infrastructure management operations to the specific environment. Whether discovering, deploying, moving, migrating, or consolidating server resources, HP OpenView Service Delivery Controller activates IT, delivering business agility, and infrastructure management to the organization.

Service Delivery Controller's powerful workflows standardize repetitive, ad hoc, and error-prone operations, improving the success of change and delivering reliable results. Pertinent settings and configurations are captured and completely adapted to the new environment for error-free, repeatable, traceable results.

Additionally, HP OpenView Service Delivery Controller encapsulates best practices into the operations it performs—improving the quality of the infrastructure over time and creating an evolutionary effect that optimizes for change and flexibility in the future.

Service Delivery Controller gets to work fast, installing right into an up-and-running environment, and actually detecting the existing infrastructure. It facilitates the newest technologies available, such as Microsoft's Automated Deployment Services (ADS), and other standard tools on which users rely.

Through its intuitive graphical user interface (GUI), the SDC organizes the complexity of server lifecycle management and file sharing services tasks, giving the user instant access to system resources, workflows, and status.

Whether the task is to utilize server capacity more efficiently, simplify server migration, navigate business expansion or consolidation, or repurpose old servers gracefully, HP OpenView Service Delivery Controller extends the physical and software infrastructures' lifecycles.

For more information on the HP OpenView Service Delivery Controller, refer to the Help documentation available at the Service Delivery Controller installation location.

Service Delivery Building Blocks

This section provides a high level overview of the building blocks of the HP OpenView Service Delivery.

Solutions

Solutions are the cornerstone of HP OpenView Service Delivery. They provide the components needed to solve particular problems, such as deploying a new server or migrating an older server to a new destination.

Solution Components

A typical HP OpenView Service Delivery solution is comprised of the following components: the Package or Packages that actually perform the operations, the UI interfacing with the user visually, and the Command-Line Interface (CLI) that allows the user to call the solution from an appropriate command prompt.

Packages

An HP OpenView Service Delivery Controller Package provides the manifestations of the logic for a solution, or, in other words, its workflow. For example, the File Sharing Services Migrate Package contains the detailed steps required (workflow) to migrate a share from source to destination. Steps, or actions, are performed by execution units (EUs) and/or other sub-packages. Thus a package is nothing more than a collection of execution units (EUs) and/or sub-packages connected in a flowchart.

A Package or a collection of Packages put together in a workflow creates a solution. For example, the File Sharing Services Migrate solution is a workflow consisting of Packages.

Package Anatomy

A Package (which consists of a collection of inputs, flow (logic), EUs, and so on) begins with a logical start. The first EU of the package takes a set of inputs, such as share name and the server name to which it belongs. Once the EU has been executed and produced an output, this output can then be used as an input to a following EU.

The entire Package can be rolled back if any part of it fails. If an EU toward the end of the Package fails, all previous EUs will roll back. Once the final EU of the Package has re-turned “true,” the Package is complete and can no longer be rolled back. If the Package has child packages nested inside of it that have completed successfully, these completed packages cannot be rolled back.

Each EU in the Package works by using its own inputs and creating an output, which the EU following may or may not consume. If any element of the Package or of any individual EU fails, the entire Package will fail. The Package and each EU in it must be registered with the repository, which will be discussed later.

Once every element in the Package has executed, the Package Designer produces several files as an output: an XML file, a C# file, and a message file.

The XML File

The XML file that is output by the Package Designer contains the layout of the shapes, so that the way the Package looks in Microsoft Visio is maintained. This information also includes the shape types that are used, the workflow layout, what individual EUs and which versions are included in the Package, and so on. Linkage information is also included.

The C# File

The Package Designer also outputs a C# file, which contains the code that actually does the Package’s work. The C# file code follows the same flow as the Package shapes in Visio; so, the code for each EU is in the same order as it is in your Visio workflow. It follows the same logic. Whereas the XML file describes the Package (layout, dependencies, inputs, outputs, and so on), the C# file contains the manifest representation of the workflow itself. When the Package is registered, you must provide your C# code as a compiled .dll file.

The Message File

There is also a message file that is output along with the XML and C# files. It contains debug messages, as well as any other messages or information the user might want to include. This message file can be extended by the user. For more information on message files and how they’re created, please see the section on Execution Units.

Working with Resources

HP OpenView Service Delivery Controller’s entire architecture is based upon resources, the primary means by which Packages are executed. Each server contains properties such as OS, server name, and so on. Under the server are also a collection of services (such as file services, IIS services, and SQL services). Underneath the level of services is a collection of file shares, which contain information such as name, path, disk space used, and so on.

The architecture of Service Delivery Controller is based on discovering this information and performing actions on these resources—this is the means by which SDD performs and completes its operations. In order for this to occur, the Package must be associated with the resource. Once it has been developed, and associated with the resource, it can be executed. Resource association provides several advantages to the Package, such as source share and source server information, and these pieces of information are provided automatically. How the Package gets attached to the resource is discussed in the following section. It is done in the XML file associated with the Package.

Binding to a Resource

In order to bind your Package to a resource, you must first make sure that the resource exists in the resource database. When you drop a resource bind statement onto the page, the Package Designer provides you with a list of every resource that is available for binding. You will not be able to bind to a Windows machine; you must instead bind to a specific instance of a Windows machine. Once this has been done, the properties of that resource will be auto-populated for you in the UI. Once you've bound to the resource, you will be able to start using EUs, such as for each statement to iterate through the shares on a server. For example, for each dependent in your share, you can iterate through the For loop and do the logic required to correct the dependency.

Managing Dependencies

One of the great advantages of the SDD is its ability to manage dependencies. The Package Designer provides information on dependencies automatically. When working with resources, you can ask, “What does this resource that I'm attaching to depend on, and who depends on it?” By dropping the resource onto the page, you can determine automatically what dependencies will be involved by using that resource—no search will be required. The Package Designer will automatically know (through reflection) what the resource's possible dependents will be.

Execution Units (EUs)

The use of Execution Units (EUs) is the primary vehicle by which HP OpenView Service Delivery carries out workflow functionality. Each EU performs a specific task or set of tasks, and a collection of EUs along with the logic that strings them together is what makes up an HP OpenView Service Delivery workflow. The SDD allows its users, among other things, to add, remove, or modify EUs in order to add robustness and flexibility to their workflows.

An EU is a unit that is agnostic of what comes before or what follows it in a Package. It is entirely independent, and acts on whatever information it calls for in order to produce its proscribed action.

The architectural concept behind EUs is that they are small atomic units of functionality, easily created, changed, and used. Therefore, it is important to remember that, in order to add new functionality to your workflow, it is advisable to create a new EU rather than adding functions to an existing EU. This would result in the EU becoming monolithic and difficult to manage. If an EU does perform more than one function, state information must be provided in order to achieve Rollback and Cleanup for each individual function.

EU Components

An EU is made up of several different files: a C# file, which contains the actual work implementation definition done by the EU; an XML file, which contains definitions of different EU portions (and, incidentally, is the part of the EU that communicates with the Package Designer); and a message file, which provides the message types that the EU will need to output. The XML file defines inputs, outputs, and state variables, as well as any other elements that appear in the Package. The C# files operate independently of the Package Designer; the EU relies solely on the XML file for its interaction with the Package Designer.

EU Part One: The C# File

There are three separate sections that make up this file. These are Execute, Rollback, and Cleanup.

- **Execute** is where the primary work of the EU is accomplished. Execute performs the specific operation of execution.
- **Rollback** is called if the Execute portion of the EU fails. It is also called if the Package that the EU is a part of fails after the EU has executed. When an EU is executed, it is placed on the Rollback stack. If the EU (or any subsequent EU) fails, recovery operations proceed. Rollback is initiated and provides, where appropriate, recovery to the original state. If no rollback is required (for example, for operations that read data), this section will simply return true.
- **Cleanup** is called if the EU operation leaves behind something that must be deleted. If no cleanup is required, this section will simply return true.

The DLL produced by the compilation of the C# file, is what is registered with the Service Delivery Controller system.

EU Part Two: The XML File

The XML file for any EU holds the responsibility for communicating with the SDD Package Designer. It contains the definitions for all the different parts of the EU, and in particular its inputs, outputs, and state information. It provides a representation of the EU to the rest of the system.

EU Part Three: The Message File

The message file for an EU must first be created in XML. The XML file specifies what different types of messages the EU can use. Following are two examples of log messages:

```
<LogMessage Id="0"
    Severity="Info"
    SymbolicName="FileServicesAddShareSuccess"
    Language="English"
    MessageFormat="Success: New share {0} added on server {1}" />
<LogMessage Id="1"
    Severity="Error"
    SymbolicName="FileServicesAddShareFailed"
    Language="English"
```

```
MessageFormat="Error: Unable to add share {0} on server {1}" />
```

Once the XML file has been prepared it will need to be converted to a C# file so that it can be compiled in with the EU. This is done with the mc2msg2cs Converter Tool that is included with the Service Delivery Controller installation. If a change needs to be made to the message file, you will make the change to the XML file, convert the XML to C# using the mc2msg2cs Tool, convert the C# to a .dll file, and so on.

Even though a message file is optional, it is recommended that it be provided with every EU, regardless of whether or not the EU needs to log any messages.

EU Registration

Each EU created must be registered with the HP OpenView Service Delivery repository. You must provide the repository with the EU .dll, XML, and message files to complete registration. EU registration is achieved by invoking the “RepositoryTool.exe” tool from the command prompt.

Capturing State Information

Any EU that is complex enough to have stages in its operation will need a place where complex information is provided, so that if a failure occurs, Rollback will have data on the “state” the EU was at when it failed. State information operates as a sort of bookmark in the operation of the EU, in case of failure. Note that state information can also be used in case of success for the EU execution Cleanup.

State information allows Rollback to function if the EU fails at some point in its own execution. Without state information in the EU, no information in it persists. This information is only of use to the EU itself—the Package Designer does not need any individual EUs state information.

It is helpful to think of the EUs being pushed onto a LIFO (last in, first out) stack as they are executed. Because of the EUs atomic nature, its original state can be restored by traveling back up the stack. This becomes necessary if the Package that the EU is a part of fails at some point past where the EU has completed execution. When Rollback of the entire Package takes place, Rollback travels back up the Package and restores the original state of each EU in order. Cleanup is called if the EUs execution is successful.

EU Creation Flow

The creation of an EU involves only a few simple steps, allowing users to easily and quickly add new functionality to Packages. For more detailed information about each of these steps, please see the sample EU sections.

- 1 Create an EU class: This is done via the C# file.
- 2 Create your EU markup file (XML).
- 3 Create your message file in XML and convert it to C# using the mc2msg2cs Tool included as part of the SDD installation. For more information on tools, please see the Tools section.
- 4 Test your EU independently using the stand-alone EU tester tool. This tool is included as part of the SDD installation. For more information on tools, please see the Tools section.

- 5 Compile your EU C# files (or class) into a .dll file: You must use a .NET development environment to compile your code. You will need the .dll files HPSD.Core.P1.dll and HPSD.PA.Developer.dll to compile your C# file. These files are included with your Service Delivery Controller installation.
- 6 Use the `RepositoryTool.exe` tool to register the EU .dll, markup, and .msg files. This tool is included as part of the SDD installation. For more information on tools, please see the Tools section.
- 7 Use the EU in a package using the Package Designer. Once you've completed the above steps, the EU that you created will be available in the Package Designer.

Execution Unit Examples

This section is a tutorial that examines two HP OpenView Service Delivery Controller Execution Units (EUs), including their accompanying XML definitions.

This tutorial provides a guide through the EU files, from the C# class definitions to the markup file definitions.

We have assumed a familiarity with the design concepts of an Execution Unit as explained elsewhere in this document.

Sample Execution Unit One: SampleEU

The first EU to be examined is a version of a “Hello World” Execution Unit, called SampleEU. SampleEU appends an input string to a specified file. For example, SampleEU could be used whenever a small amount of information needs to be added to a file in the database. In this case, SampleEU will add the text “Hello World!” to the file that is specified.

This sample shows how Execution, Rollback, and Cleanup work. State information is not required in this case, because this EU does not have any information that needs to be persisted.

The XML definition file associated with SampleEU contains definitions of all of its elements, including string definitions. The code for both the C# and XML files is shown in Appendix A in listings A-1 and A-2 respectively.

SampleEU Class Elements

The first thing that happens in this EU is that the namespaces are defined. The definition of all of these namespaces is provided here as a convenience, in order to prevent having to spell out the namespace at each individual spot in the file where it would be required later.

```
using System;  
using System.IO;
```

```
using Microsoft.Win32;  
using HPSD.PA.Developer;  
using HPSD.Core.P1;  
namespace HPSD.Solutions.EU.Samples.Impl
```

The first real action that is taken in the file is that the interface execution unit (IEU) is called upon. All EUs inherit from this interface. By doing so the EU must implement all of the interfaces that IEU requires, namely, Execute, Rollback, and Cleanup, the three basic parts of an EU, which were discussed earlier. This EU also contains 5 private strings, and one Boolean string.

```
public class SampleEU : IEU  
{  
    private string    m_strInput = "";  
    private string    m_strFileName = "";  
    private string    m_strDirectory = "";  
    private string    m_strMachineName = "";  
    private string    m_strTempFileName = "";  
    private bool      m_bWasCreated = false;
```

The first step is to provide the string that will be input:

```
public string Input  
{  
    get  
    {  
        return (m_strInput);  
    }  
    set  
    {  
        m_strInput = value;  
    }  
}
```

Then add the machine name where the file that the string will be added to resides:

```
public string MachineName  
{  
    get  
    {  
        return (m_strMachineName);  
    }  
    set  
    {  
        m_strMachineName = value;  
    }  
}
```

Next, we must know whether the file was created or not:

```
public bool WasCreated  
{  
    get  
    {
```

```

        return (m_bWasCreated);
    }
    set
    {
        m_bWasCreated = value;
    }
}

```

Now we need the name of the file where the string will be appended:

```

public string FileName
{
    get
    {
        return (m_strFileName);
    }
    set
    {
        m_strFileName = value;
    }
}

```

Then the name of the directory that the file is in:

```

public string Directory
{
    get
    {
        return (m_strDirectory);
    }
    set
    {
        m_strDirectory = value;
    }
}

```

Lastly, we need the temporary file name, so that Rollback can be achieved:

```

public string TempFileName
{
    get
    {
        return (m_strTempFileName);
    }
    set
    {
        m_strTempFileName = value;
    }
}

```

Now that all of the above information has been provided, we execute the EU. Execute copies the original file to a temporary file, then adds the string that was specified to the original file.

```

public bool Execute()
{
    string strFile = FullFileName();
}

```

```
m_strTempFileName = Path.GetTempFileName();

try
{
    StreamWriter w = null;
    if (!File.Exists(strFile))
    {
        // Create a file to write to.
        w = File.CreateText(strFile);

        if (w != null)
        {
            WasCreated = true;
        }
    }
    else
    {
        File.Copy (strFile, m_strTempFileName, true);
        w = File.AppendText(strFile);
    }
    if (w == null)
    {
        return false;
    }

    w.Write (Input);
    w.Close ();
}
catch
{
    return false;
}

return true;
}
```

Once Execute has successfully run, we need Cleanup, to delete the temporary file that the EU created.

```
public bool Cleanup()
{
    try
    {
        if (WasCreated)
        {
            File.Delete (m_strTempFileName);
        }
    }
    catch
    {
        return false;
    }

    return true;
}
```

Rollback is also required, to return the file to its original state if the EU fails. It does this by copying the temporary file to the original file.

```
public bool Rollback()
{
    string strFile = FullFileName ();
    try
    {
        if (WasCreated)
        {
            File.Delete (strFile);
        }
        else
        {
            File.Copy (m_strTempFileName, strFile, true);
            File.Delete (m_strTempFileName);
        }
    }
    catch
    {
        return false;
    }

    return true;
}
```

Finally, the file name's full path is built:

```
private string FullFileName ()
{
    string strFile = null;

    if (MachineName.Length == 0)
    {
        strFile = Path.Combine (Directory, FileName);
    }
    else
    {
        string tmp = Directory;
        tmp.Replace (':', '$');
        tmp = string.Format ("\\\\\\{0}\\\\{1}", MachineName, tmp);
        strFile = Path.Combine (tmp, FileName);
    }

    return strFile;
}
}
```

Sample Execution Unit Two: AddShare

The EU AddShare is a FileServices EU that is used to add a share (without permissions) to a specified destination that has been given a server name and a share information directory.

This sample shows how Execution and Rollback work, as well as including information on creating state information. This EU example does not require Cleanup.

The XML file associated with the EU contains definitions of all of the AddShare elements, including string definitions.

The XML definition file associated with AddShare contains definitions of all of its elements, including string definitions. The code for both the C# and XML files is shown in Appendix B in listings B-1 and B-2 respectively.

AddShare EU Class Elements

At the top of the AddShare C# file, it tells us that it's using these namespaces, as well as the namespace that the EU occupies. For example, HPSD.PA.Developer provides the IEU method information (see the next paragraph for more information on IEU).

```
using System;
using System.IO;
using Microsoft.Win32;
using HPSD.PA.Developer;
using HPSD.Core.P1;
using HPSD.Win32.Netapi32;

namespace HPSD.Solutions.EU.FileServices.Impl
{
```

The first real action that is taken in the file is that we call upon the method IEU. All EUs inherit from this method. By doing so the EU must implement all of the methods that the IEU requires, namely, Execute, Rollback, and Cleanup, the three basic parts of an EU, which were discussed earlier. The namespace for IEU is the HPSD.PA.Developer namespace.

```
public class AddShare : IEU
{
```

Here, the first string is defined, which happens to be mServerName. This string specifies the name of the remote server on which the function is to execute. The string must begin with \\ . If this parameter is NULL, the local computer is used. It is important to remember to prepend (prefix) the preceding slashes to the ServerName.

```
private string mServerName = null;
    public string ServerName
    {
        get
        {
            return mServerName;
        }
    }
}
```

```

    }
    set
    {
        mServerName = value;

        if (mServerName != null)
        {
            if (!mServerName.StartsWith("\\\\"))
            {
                mServerName = "\\\\" + mServerName;
            }
        }
    }
}

```

Next, we call the buffer that specifies the share metadata:

```
private NetShare.ShareInfo2 mShareInfo = null;
```

Then we call the property used to get and set the share information:

```
public NetShare.ShareInfo2 ShareInfo
{
    get
    {
        return mShareInfo;
    }
    set
    {
        mShareInfo = value;
    }
}

```

Now, we provide some information that will be needed for Rollback and Cleanup. At this point we are ready to start setting state information. State information allows for information in the EU to persist. From the comments included here, you can see that the execution state provides a status on where the execution is, at any particular point in time. This way, if there is a problem with the EUs execution, state information allows you to go back and know exactly where the problem occurred. So, in the example provided here, the AddShare execution is begun, but the state information provides for a different action to be taken if, for example, the share to be added already exists.

First, we need execution status:

```
private ExecState mExecStatus = ExecState.Begin;
```

Then, we need the property accessor for the execution status and state:

```
public ExecState ExecStatus
{
    get
    {
        return mExecStatus;
    }
    set
    {
        mExecStatus = value;
    }
}

```

```

public enum ExecState : int
{
    /// <summary>
    ///     Begin Execution
    /// </summary>
    Begin          = 0,

    /// <summary>
    ///     Share Already Exists
    /// </summary>
    ShareExists    = 1,

    /// <summary>
    ///     Share Added
    /// </summary>
    ShareAdded     = 2,

    /// <summary>
    ///     End Execution
    /// </summary>
    End,
}

```

Now, we get into the section that describes the three primary portions of this (and any) EU: Execute, Rollback, and Cleanup. In the Execute method, the work of the AddShare EU is accomplished. The EU must validate the candidate share, add the share, then set the state that the share has been added and log that message. If the share does not validate, or if the share already exists, then Execute returns failure and an error is logged that the AddShare has returned failure.

```

public bool Execute()
{
    // Enter
    mEuLoader.Logging.SubmitFunctionEnter();

    // Assume failure
    bool retval = false;

    // Set Execution Status
    SetExecStatus(ExecState.Begin);

    // Validate the share to be added.
    if (!NetShare.Exists(mServerName, mShareInfo.NetName))
    {
        // Add the Share;
        if (NetShare.Add(mServerName, mShareInfo))
        {
            // Share Added
            SetExecStatus(ExecState.ShareAdded);

            // Log the Message
            mEuLoader.Logging.SubmitOperatorLog(
                LogMessage.FileServicesAddShareSuccess,
                mShareInfo.NetName, mServerName);

            retval = true;
        }
    }
}

```



```

else
{
    // Log the Error
    mEuLoader.Logging.SubmitOperatorLog(
        LogMessage.FileServicesAddShareFailed,
        mShareInfo.NetName, mServerName);
}
}
else
{
    // Share Exists;
    SetExecStatus(ExecState.ShareExists);

    // Log the Error
    mEuLoader.Logging.SubmitOperatorLog(
        LogMessage.FileServicesAddShareExists,
        mShareInfo.NetName, mServerName);
}

// Leave & Return
mEuLoader.Logging.SubmitFunctionLeave(retval);
return retval;
}

```

This section provides the necessary Rollback method, which is called if a failure occurs (during this or subsequent EUs in the package). Rollback deletes the share that has been added. It then logs that the rollback was successful, or logs that an error occurred if the rollback failed.

```

public bool Rollback()
{
    //Enter
    mEuLoader.Logging.SubmitFunctionEnter();

    // Assume success
    bool retval = true;

    // If the Share has been added then remove it
    if (mExecStatus == ExecState.ShareAdded)
    {
        // Delete the Share;
        if (NetShare.Delete(mServerName,
mShareInfo.NetName))
        {
            // Log that it was successful
            mEuLoader.Logging.SubmitOperatorLog(
                LogMessage.FileServicesAddShareRemoved,
                mShareInfo.NetName, mServerName);
        }
    }
    else
    {
        // Log the error
        mEuLoader.Logging.SubmitOperatorLog(
            LogMessage.FileServicesAddShareRemoveFailed,
            mShareInfo.NetName, mServerName);

        retval = false;
    }
}

```

```
        }  
    }  
  
    // Leave & Return  
    mEuLoader.Logging.SubmitFunctionLeave(retval);  
    return retval;  
}
```

This section looks at Cleanup. In the AddShare EU, Cleanup is not required, so it just returns true.

```
public bool Cleanup()  
{  
    return true;  
}
```

The final section of this EU file takes care of one last step, namely setting the Rollback status.

```
private void SetExecStatus(  
    ExecState Status)  
{  
    mExecStatus = Status;  
    mEuLoader.SetRollback();  
}  
}
```

SDD Tools

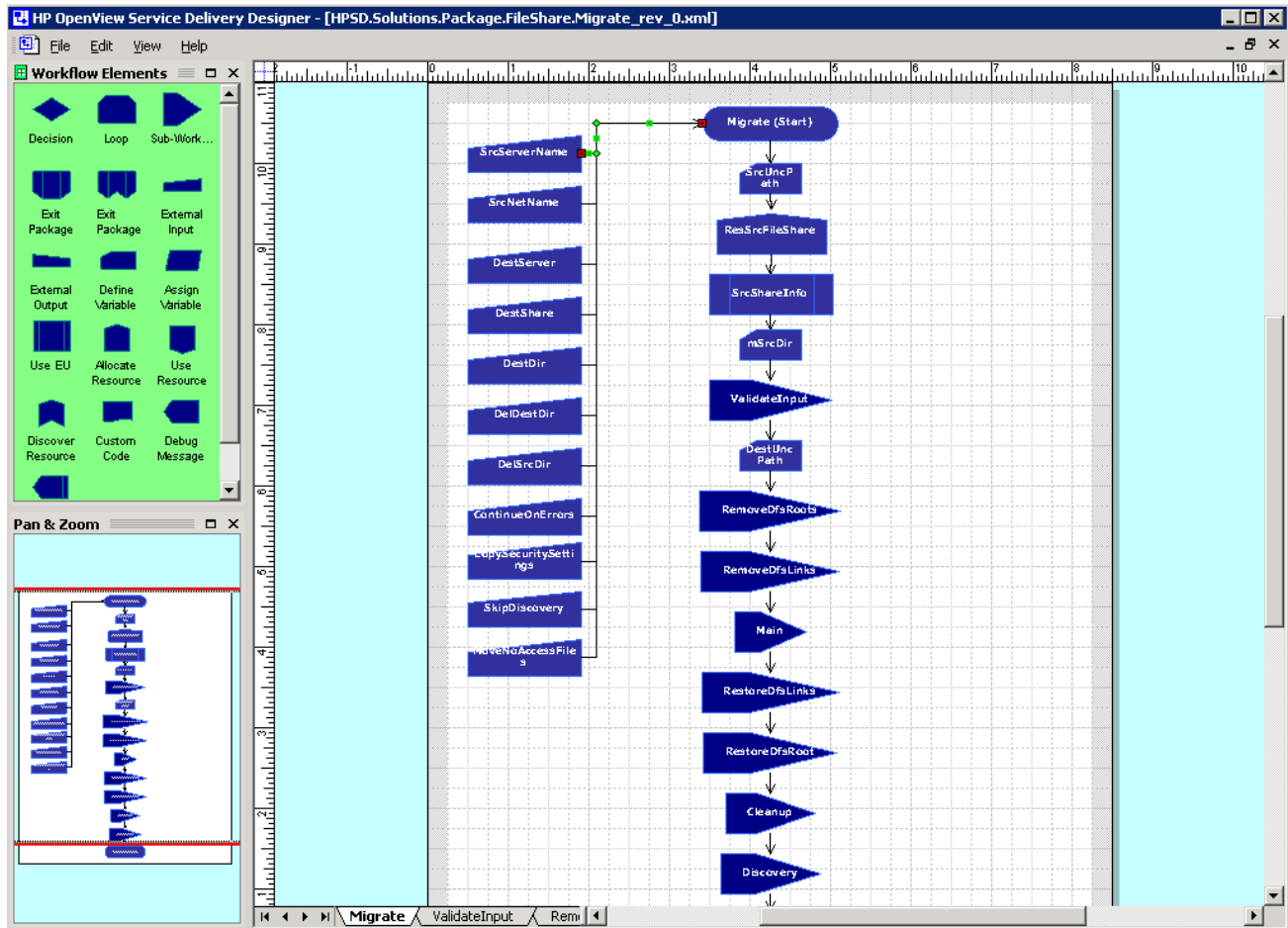
The Package Designer

This section walks you through using the Package Designer tool, which is the heart of the HP OpenView Service Delivery Designer.

Overview of the Package Designer

The Package Designer Tool lies at the heart of the HP OpenView Service Delivery Designer. The Package Designer provides a simple tool to capture and customize an IT organization's workflow. The greatest advantage of using the Package Designer is that it allows you to insert new functionality, such as EUs, logic, or sub-workflows (child packages), into existing workflows. Rather than using a collection of scripts, the Package Designer provides a representation of the code that can be used to create a workflow using individual atomic units of functionality called Execution Units (EUs).

The Package Designer also provides GUI and CLI customization options for solutions.



An execution unit (EU) is an atomic vehicle that includes one or more inputs that produce one or more outputs; that does not call for UI to be displayed, nor have any side effects. It does not require anything to be run before it, nor leaves any “bread crumbs” after it for other EUs that may follow. For further information on EUs, go to the section on Execution Units.

SDD includes a set of existing packages, as well as a collection of EUs that will allow you to modify these packages to suit your needs. When you open the Package Designer, you will select from a collection of existing packages. The product allows you to both view and update a workflow. You will also be provided with the tools to create your own EUs for modification of the packages.

Launching

On the server where SDD is installed, go to the Programs group and launch the Package Designer from the program menu, following the HP OpenView Service Delivery menu item and selecting the SDD link.

You can also launch the Service Delivery Designer’s Package Designer by loading the PackageDesigner.vsd file into Microsoft Visio 2002. The PackageDesigner.vsd file is available at

C:\Program Files\HP\ServiceDelivery\Designer

Assuming the SDD has been installed at C:\Program Files\HP\ServiceDelivery\Designer.



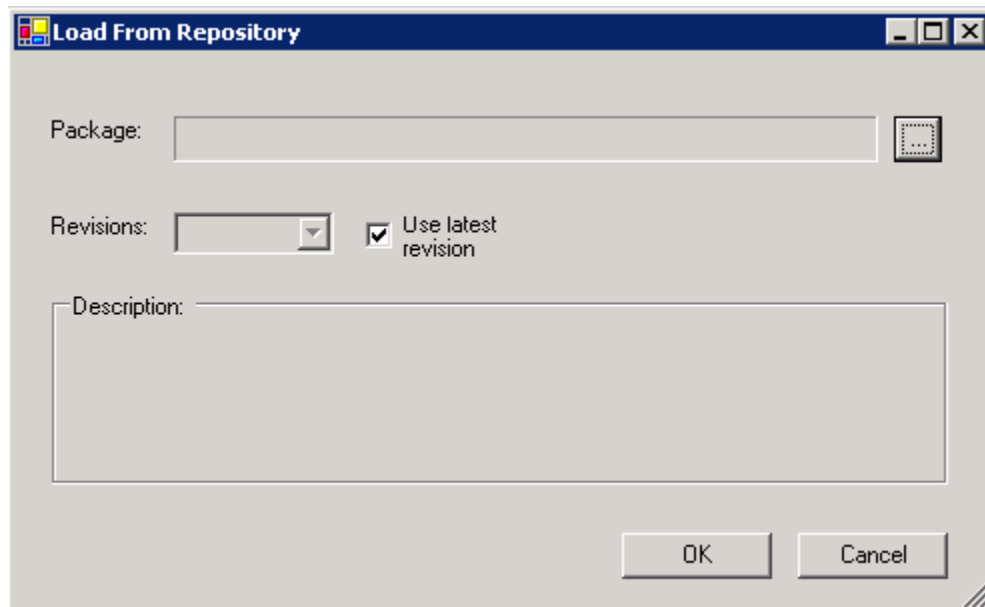
You must already have Microsoft Visio 2002 installed on the machine where Solution Builder is installed. Only Microsoft Visio 2002 works with the Package Designer.


The Package Designer is now running and ready to open Service Delivery Controller packages.

Tip: it is very useful to have Microsoft Visio's "Pan & Zoom" window tool available. Before starting the Solution Builder, start Microsoft Visio 2002, create a new blank Visio document and in the View menu select Pan & Zoom Window. Then launch the Package Designer as described here.

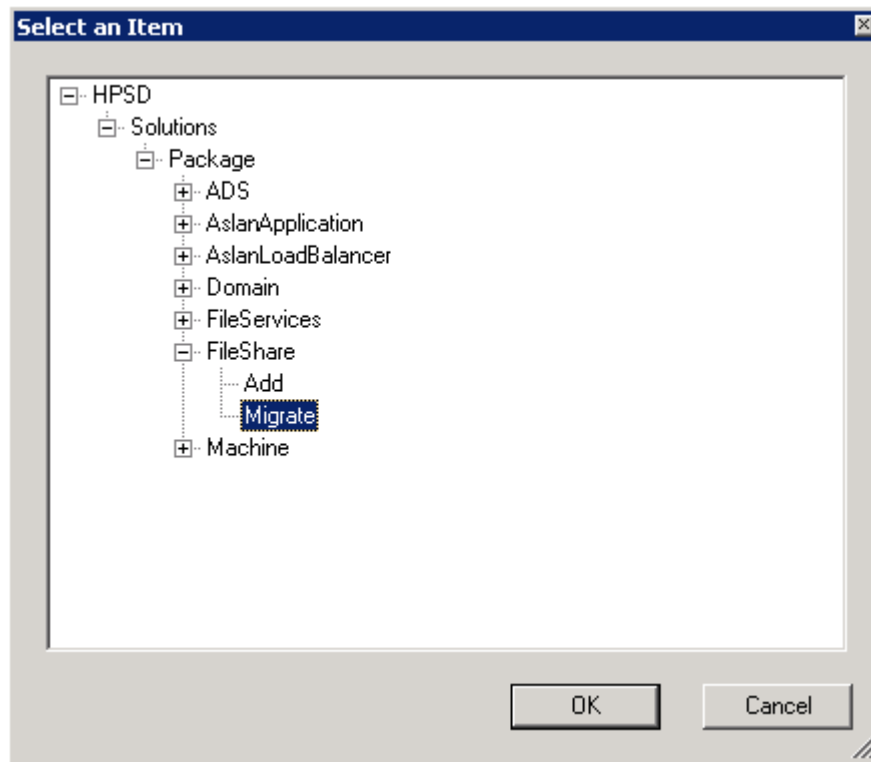
Opening a Package

To open a package, select the Load From Repository command under File. The following dialog comes up:

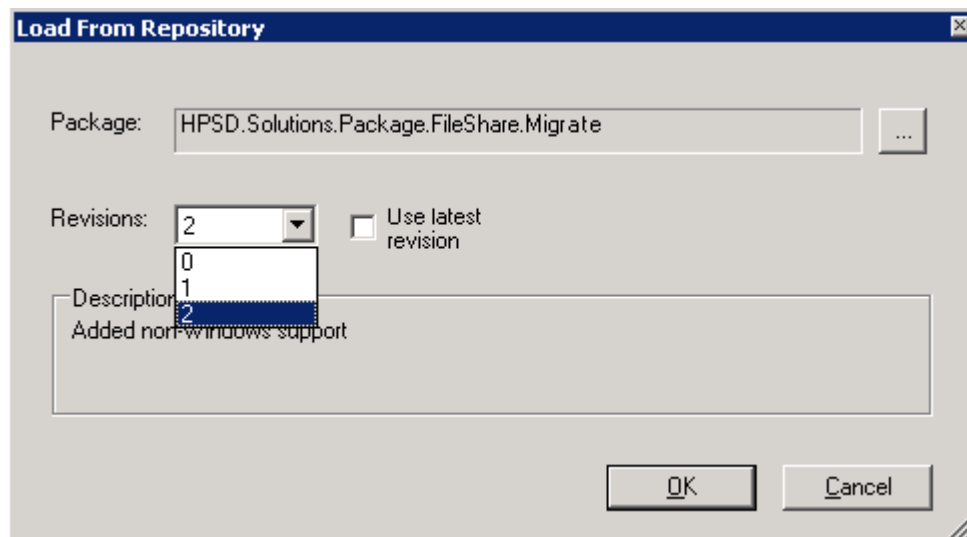


Next, select the Package you want to edit. To do so click the  button.

The following dialog appears:



In this example the user has elected to load the FileShare Migrate Package. Click **OK**. A dialog similar to the following appears:



The Package Designer provides the option to edit any of the Package revisions. In the example shown here the user has already de-selected the “Use latest revision” option, thus enabling the revision selection dropdown. In this example the Migrate Package is at revision 2 (in other words, there are three revisions in the system of this Package, 0—2). Note that when initially installed all packages are at revision 0.

Select the revision you want to edit and click **OK**. Or if you intend to edit the latest revision, leave the “Use latest revision” option checked.

The revision description is shown in the Description field, and the revisions displayed in the above dialog will vary based on the revision history of a Package.

Saving a Package

After you have finished editing a Package, select the Save to Repository option from the Package Designer File menu. The following dialog appears:

Provide the description you want for the Package and click **OK**.



You must provide a description in order to save the package.

The revision number is automatically determined by the host HP OpenView Service Delivery Controller.

Saving a Package Locally

You may save a package locally (on the disk) as opposed to the Service Delivery Controller Repository. Doing so is convenient during the package development process. Note however, that saving the package locally does not update the Service Delivery Controller repository. You must save the package in the Service Delivery Controller repository in order for it to be available for execution.

Note that in order to save a package locally you must first load it from the Service Delivery Controller Repository.

Saving the package locally does not affect the revision sequence of this package. Only when a package is saved in the Service Delivery Controller Repository its revision number is incremented.

Editing a Package

To change an HP OpenView Service Delivery Package revision, open it in the Package Designer. The next step is to identify the change you would like to perform. For example, you may want to modify the logic of the Package, add or remove EUs, change job options, and so forth.

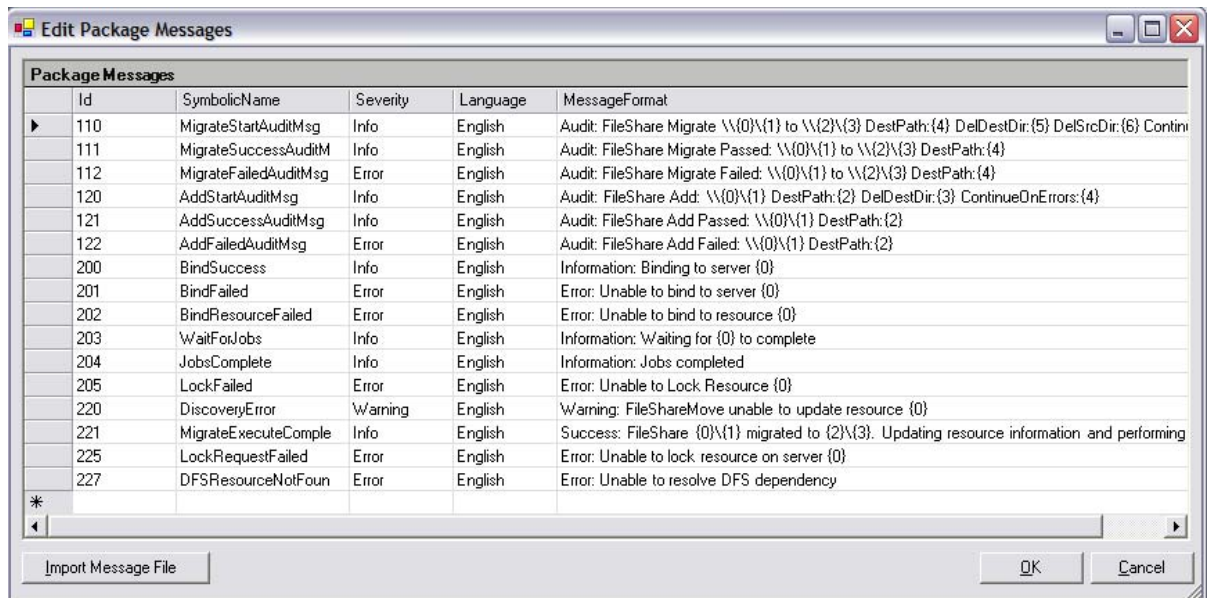
To edit a Package, identify the page (sub-workflow) you would like to edit. You can either right-click the appropriate Scope shape and select GoTo, or you can directly click the desired page from the Visio workspace as shown in the following diagram, where the user has selected the ValidateInput page:



You are now ready to perform edits to the Package. Consult the Walkthrough section for Package modification examples.

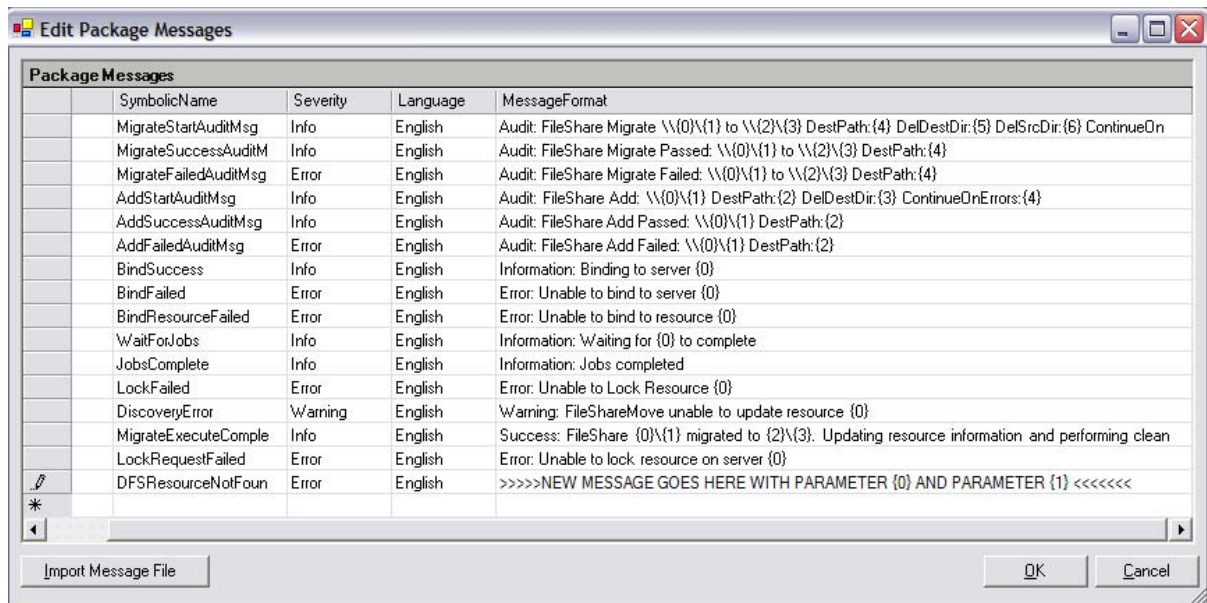
Edit Log Messages

The Edit Log Messages Tool is another powerful feature of the Solution Builder. When a message logged by the Package needs to be changed it can be done from the Action/Edit Log Messages menu. Doing so launches the Log message editor. The following image shows the message editor for the FileShare Migrate Package:



To modify the log message file, how to use message parameters must be understood.

A message can be directly edited in the Message Format column as shown below. In this example the last message in the window has been edited.



Import Message File

The Import Message File option allows you to import a message file to replace the existing message file, instead of editing each message individually. You will need to construct the message file per SDD specifications which are available from the SDD support web site.

Preview

The Package Designer provides the option to preview the code generated. To do so, select the View/Preview menu option. This causes the Package Designer Preview window to pop up. The C# code generated will be as shown in the following image:

The screenshot shows a window titled "Package Preview" with a toolbar containing three options: "C# Code" (selected), "Xml", and "Display Differences". The main area displays the following C# code:

```

//
// generated by the HP OpenView Service Delivery Designer Representation on
// 3/25/2004 10:28:50 AM
//
namespace HPSD.Solutions.Package.AslanApplication.Impl
{
    /// <summary>
    ///     The Rescale package.
    /// </summary>
    public class Rescale : HPSD.PA.Developer.PackageBase
    {
        private string m_ApplicationId;

        private string m_HealthCriteria;

        /// <summary>
        ///     Gets or sets the value of ApplicationId.
        /// </summary>
        public string ApplicationId
        {
            get
            {
                return m_ApplicationId;
            }
            set
            {
                m_ApplicationId = value;
            }
        }
    }
}

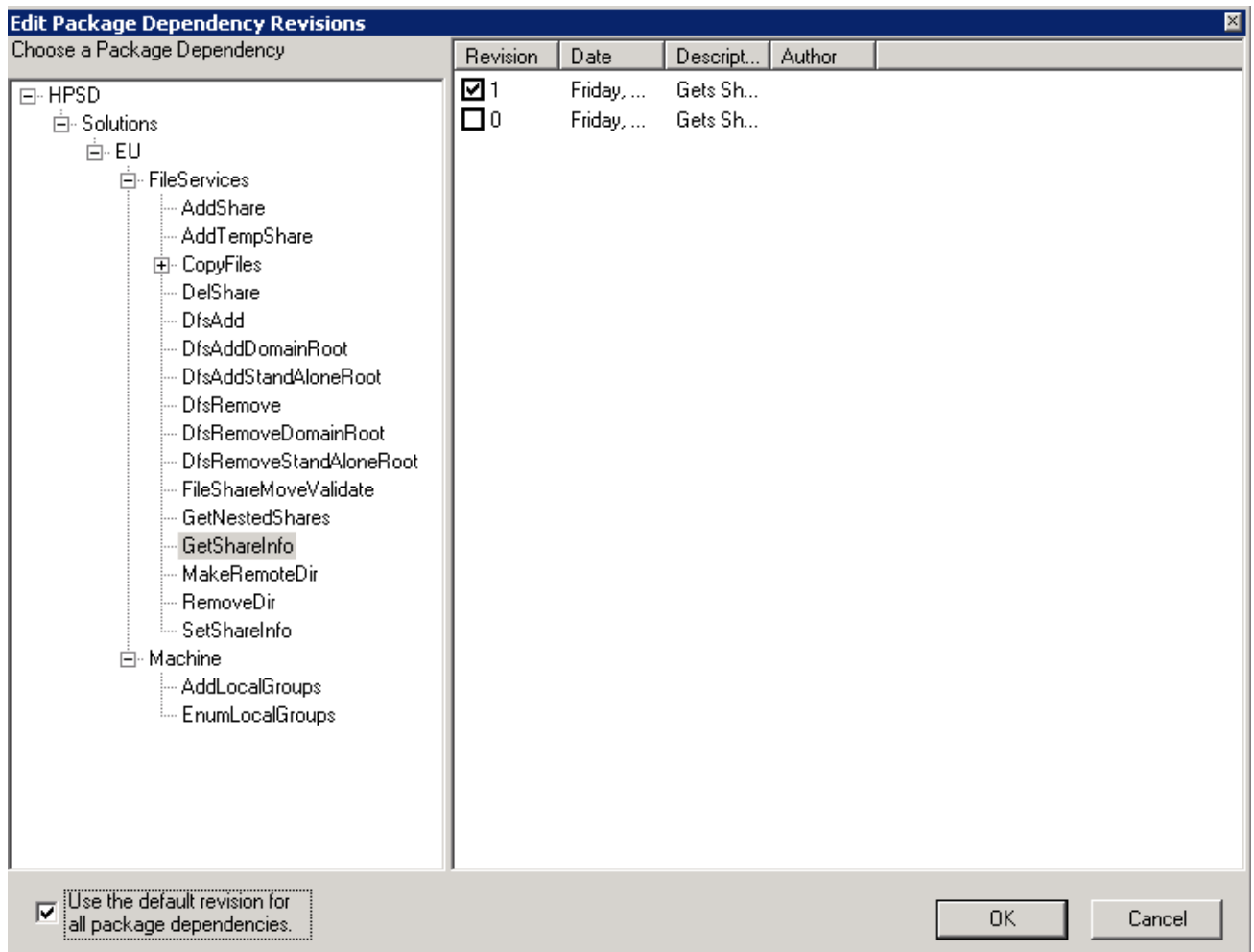
```

The XML output generated by the Package can be viewed by selecting the XML option.

The Display Differences option shows the changes made to the Package before it was saved to the repository. Differences can be viewed for both C# and XML and are indicated by the - and + signs.

Updating a Package to Latest EU Revisions

You can update a Package to the latest EU revisions. This is achieved by opening the Package in the Package Designer, selecting “Edit/Package Dependency Revisions”, and then selecting the “Use the default revision for all package dependencies.” option in the Edit Revisions dialog as shown below:



As shown here, the package will get the latest revisions in the repository. See the Versioning section of this documentation for more details on versioning and revisions.

Package Designer Shapes

When modifying SDD workflows you can use the following shapes in the Package Designer:

- **Decision (Conditional):** Adds a decision branch. Decision branches can be nested.
- **Loop:** Adds a loop. Loops can be nested.
- **Sub-Workflow:** Allows for the inclusion of a sub-workflow from one page to another. Sub-workflows can be nested.
- **Exit Package Success:** Allows the user to exit in the middle of the Package, causing the Package to succeed.
- **Exit Package Failure:** Allows the user to exit in the middle of the Package, causing the Package to fail.
- **External Input:** Adds an external input to the Package.

- **External Output:** Adds an external output to the Package.
- **Define Variable:** Defines and optionally assigns a value to a new variable in the Package.
- **Assign Variable:** Assigns a value to a previously defined variable.
- **Use EU:** Gives the option to select, add, and use an EU in the Package workflow.
- **Allocate Resource:** Adds a new resource instance into the Package workflow.
- **Use Resource:** Allows an operational method off a resource to be invoked. This is the method to use to invoke sub-packages from the current package.
- **Discover Resource:** Invokes discovery for a resource instance.
- **Custom Code:** Allows the inclusion of custom code into the Package workflow.
- **DebugMessage:** Provides user with debug message capability.
- **Console Message:** Displays a message on the Service Delivery Controller Console.

Versioning Overview

This section describes the versioning feature of the product.

Introduction to Versioning

In a typical solution creation process, many revisions (versions) are created and tested by the solution developers before the revision that passes the exit criteria is declared the “released revision.” The released revision is to be used by the operators of Service Delivery Controller for the particular solution.

As time goes by, additional changes may be needed. The above process is repeated and a new released revision is made available. By design, this new revision becomes the default version available to the Service Delivery Controller system.

If, for whatever reason, (for example, due to test or legal requirements), there is a need to run a revision other than the default revision, this can be easily achieved by invoking the desired revision number of a given solution.

In this section versioning is defined as the process of creating revisions (for EUs and Packages).

Revisions are Persistent

SDD supports versioning at the Execution Unit (EU) and the package level, and (since a solution is a collection of one or more packages) to the solution level as well.

For both EUs and Packages, when a new revision is created, it is saved as new in the Service Delivery Controller Repository and does not override the previously saved revision. For example, when an EU is saved five times, there are five versioned copies of this EU in the repository. The Service Delivery Controller repository uses a numbering scheme to keep track of revisions. The first revision gets number 0 and each subsequent revision gets the previous revision number + 1.

The same applies for Packages. When they are saved, a new copy is made and the previous one is still available.

► It is important to note that a Package saves with it the version of the EUs it uses. Thus even if the EU changes, any saved packages will run with the EU version they were last saved with. This property is called immutability.

► Important: If a package uses an EU more than one times, in the package proper, this EU must be in the same revision throughout. In other words, you cannot use one revision of a given EU at some point in the package and another revision at another place in the same package.

► However, if a package invokes sub-packages, these sub-packages may use different revisions of this EU.

Packages are immutable, in other words, they save with them the versions of the EU they use.

Package Default Revision

By design, when a revision (version) of a package is saved it becomes the default revision for this package. This is the revision that members of the ServiceDeliveryOperators will execute when they issue jobs based on this package.

By design the default revision is the latest saved Package revision. However, a member of the ServiceDeliveryAdministrators or Architects can change the default to any of the registered revisions simply by resaving the desired revision, thus creating a new revision, which becomes the default (by design).

► The default revision (version) of a Package is always the latest saved.

Changing the default revision affects only newly created jobs (jobs created after the change).

► When the default Package changes, any completed, currently running or scheduled (for the future) jobs are **not** affected. Only jobs created after this change takes effect are affected.

EU Default Revision

The Package Designer of the SDD uses the default revision of an EU. By design the EUs default revision is the latest saved in the repository.

Unless specified otherwise, the EUs default revision is the one used when this EU is added to a Package, or when the user decides to update one or all EUs of a Package to their default revisions.

Users can select any revision of an EU to participate in a Package. They are not restricted to using the default revision.

A member of the ServiceDeliveryAdministrators or Architects can change the default to any of the registered revisions simply by re-registering the desired EU revision, thus creating a new default revision, or they can register a brand new revision of the EU.

Changing the default revision affects only newly edited Package revisions (packages saved after the change).

▶ When the default EU changes, any existing packages and any completed, currently running or scheduled (for the future) jobs are not affected.

▶ Important: If a package uses an EU more than one times, in the package proper, this EU must be in the same revision throughout. In other words, you cannot use one revision of a given EU at some point in the package and another revision at another place in the same package.

Note however, that if a package invokes sub-packages, these sub-packages may use different revisions of this EU.

Propagating an EU Update to a Package

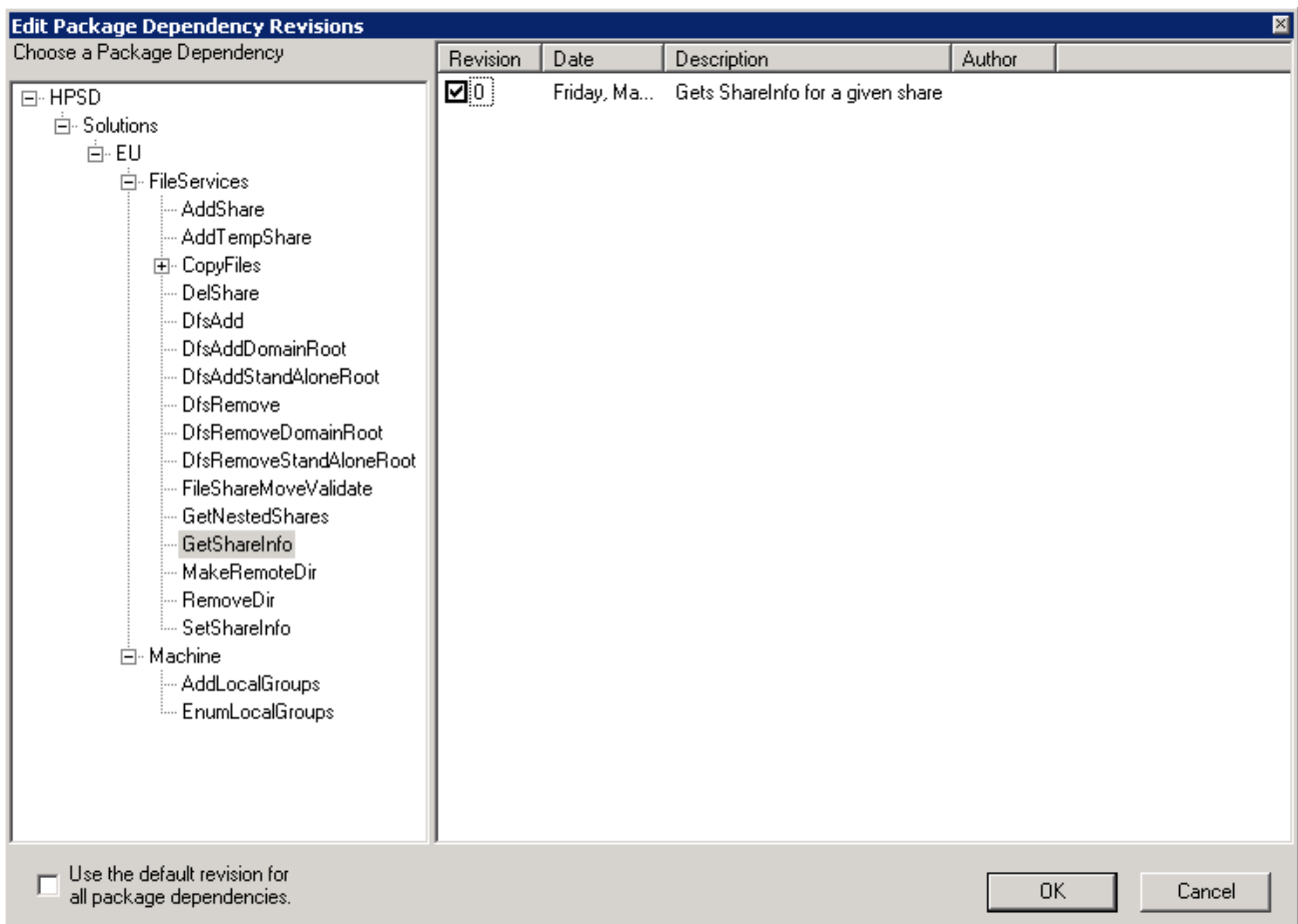
Due to the immutability property, Packages are self-contained and are not affected by any EU changes. When a changed EU needs to be propagated to an existing Package, this task needs to be performed as follows.

The new EU is registered in the Service Delivery Controller repository (using the EU registration tool as described in this documentation). If this is a brand-new EU, it creates revision 0. If this is an already existing EU, then the EU gets a revision number equal to the previous revision number + 1.

The desired Package is opened in the Package Designer.

Using the Package Designer's "Edit Revisions" feature (Edit/Package Dependency Revisions menu), find the desired EU, in the package, to update.

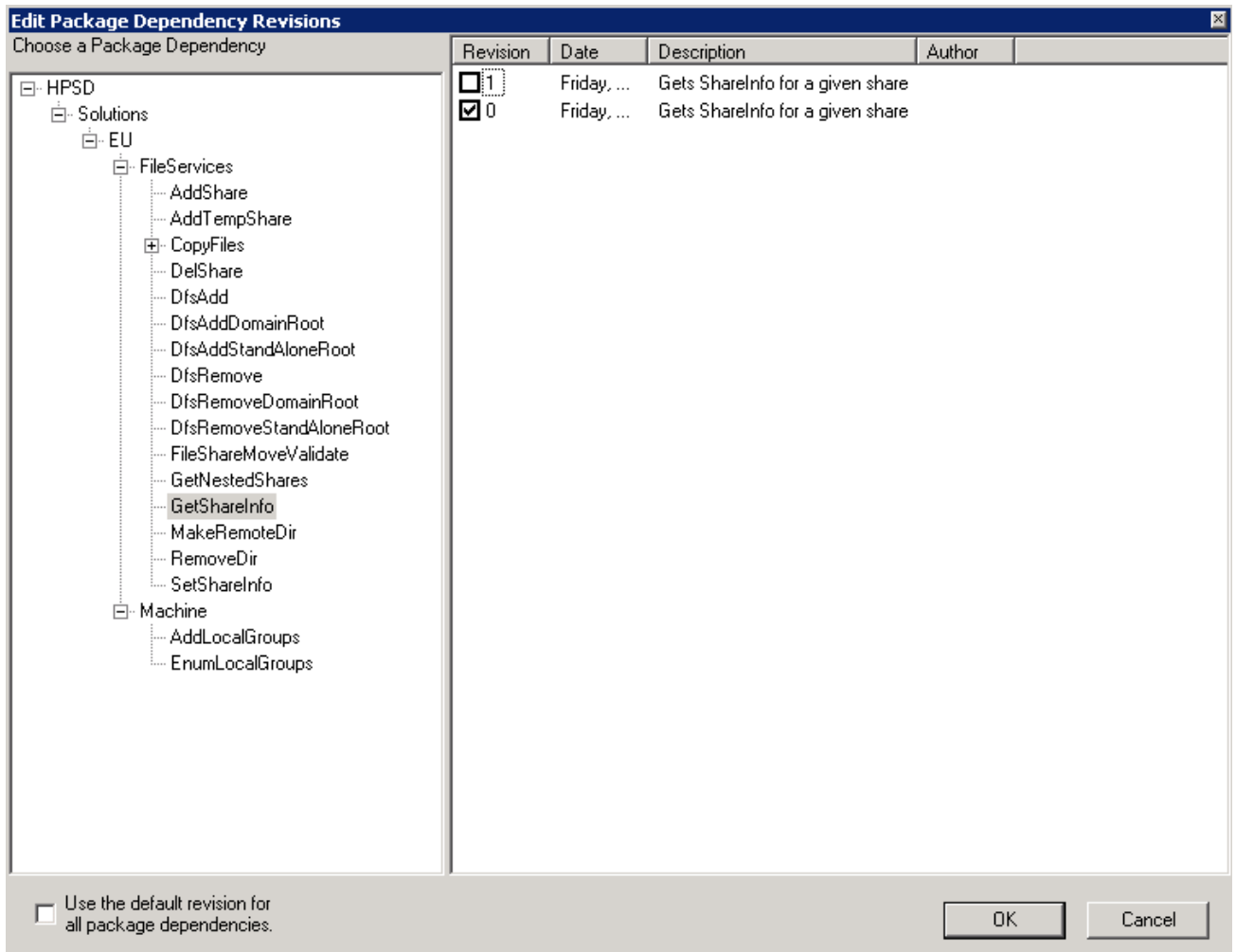
The image below shows the Edit Revisions Dialog of the Package Designer, showing all the EUs the package contains. Because the GetShareInfo is selected, the revision details are shown for that particular EU. In this case the GetShareInfo EU is at revision 0 (for example, the EU is as registered by the initial Service Delivery Controller installation).



The following image shows the same package version after a new revision of the GetShareInfo EU has been registered in the repository. The GetShareInfo EU now has two revisions, 0 and 1.

By design the default revision is revision 1 because it is the latest saved. However, the package is using revision 0 as indicated by the check box:

0 in the diagram below.

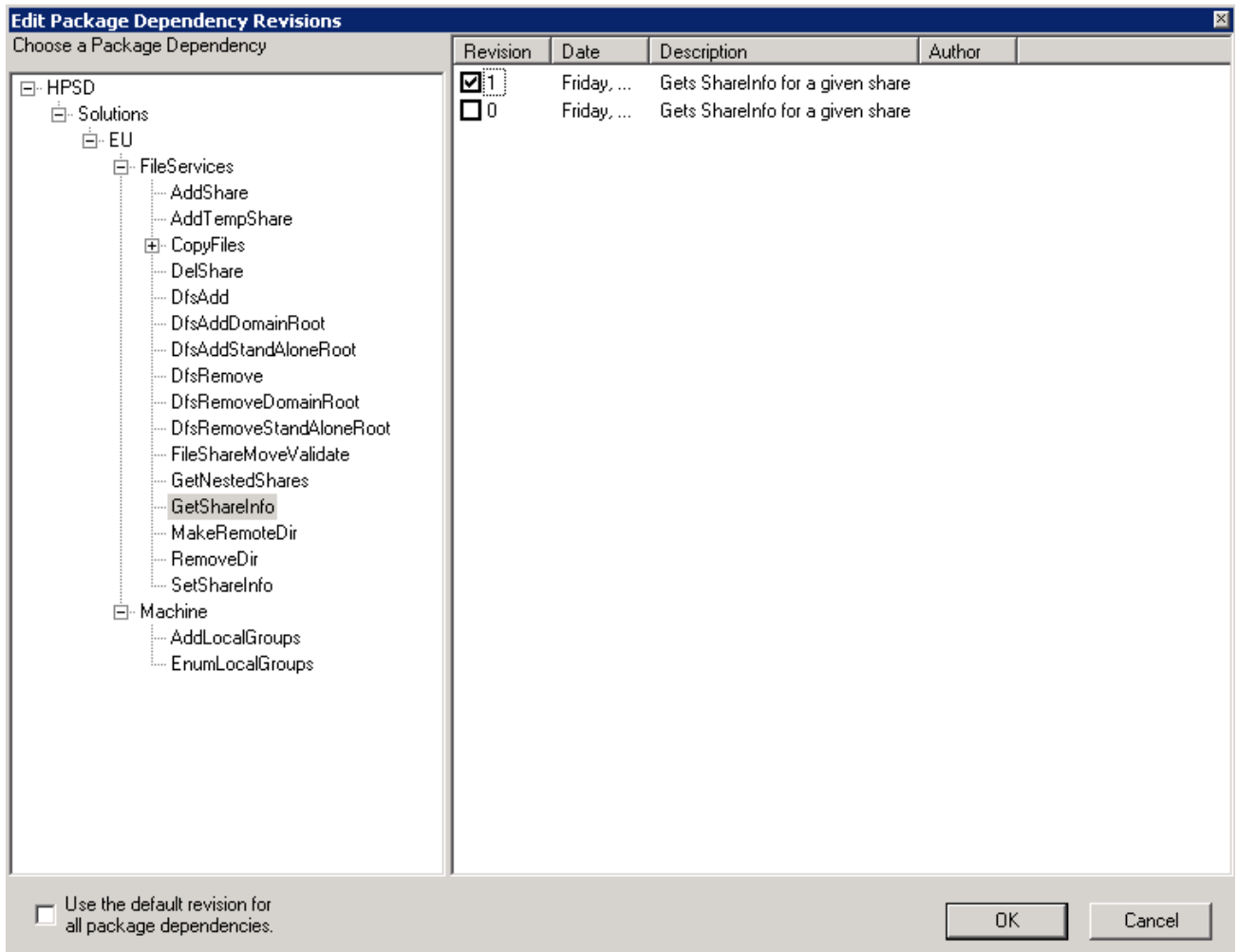


Note that the default revision need not be explicitly indicated (visually) in the Package Designer since it is always the one with the highest revision number (the one most recently saved).

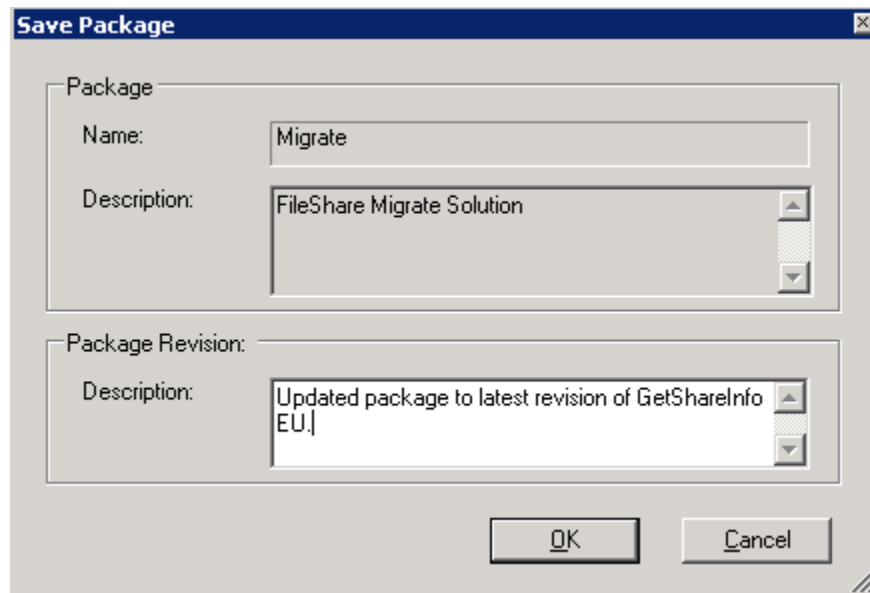


The package is still using revision 0 of the GetShareInfo EU. The Package Designer simply informs the user that additional EU revision(s) are available.

To change to the default and latest revision of the GetShareInfo EU, the revision is selected as shown in the following image.

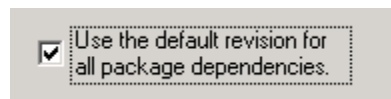


To save the Package to the repository select the Save to Repository option, enter a description of the change, and click **OK** as shown in the following image.

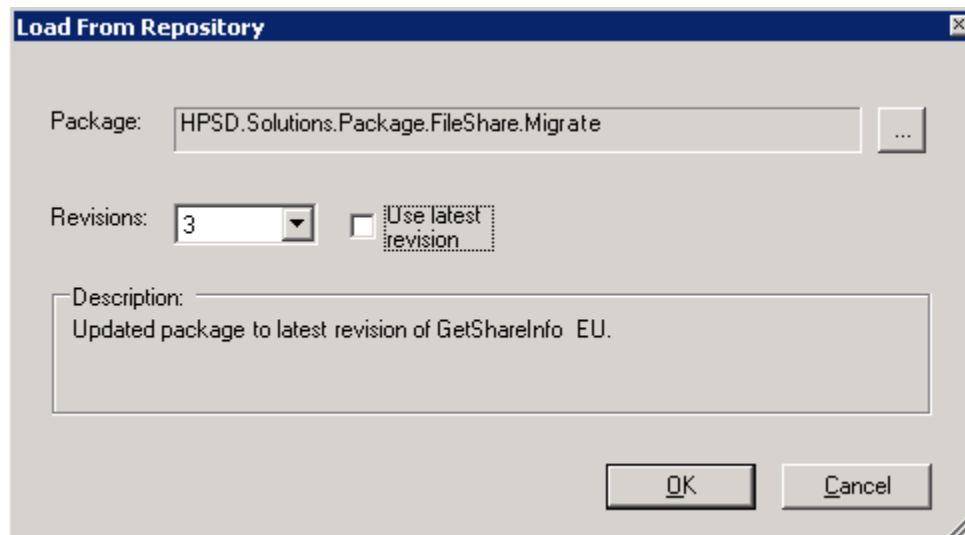


When saved, a new Package revision is created and becomes the default revision. The previous revision is available in the repository.

If all the EUs of a Package must change, the Package Designer supports this feature by selecting the “Use the default revision for all package dependencies” option in the Edit Revisions dialog as shown below.



The following image shows that when the FileShare Migrate Package is loaded again from the repository, the option to select, either the default revision or an explicit revision, is presented to the user. In this case the user has unchecked the “Use latest revision” option and has selected revision 3 of the package, which is the revision that was created when the update to the revision 1 of the GetShareInfo EU was made (do not confuse EU revision numbers with package revision numbers; they are not related).



A Versioning Example

Package P1 is in revision 8. Package P1 uses EUs E1 and E2. EU E1 is in revision 12 and E2 in re-vision 3.

When the user loads P1 in the Package Designer and then saves P1 to the repository, a new package revision is created, revision 9 in this case. This becomes the new default revision. A new revision is created even though nothing in the package has changed.

Assume now that new revisions of EUs E1 and E2 are available, the desired revisions (for this package) being 17 for E1 and 4 for E2 (note here that due to multiple registrations the desired revision of EU E1 is not number 13 but number 17, because EU E1 has been modified 5 times). To select these revisions of EUs E1 and E2 to be used in the package, the user has two options. Either explicitly select the desired revisions of EUs E1 and E2 and then save the package, or elect to update the package to the latest EU revisions available in the system.

In either case when package P1 is saved to the repository it will be in revision 10 with EU E1 in revision 17 and EU E2 in revision 4. However, if the latter method is selected, any additional EUs (other than E1 and E2) in the package P1 will also be updated to the default revision.

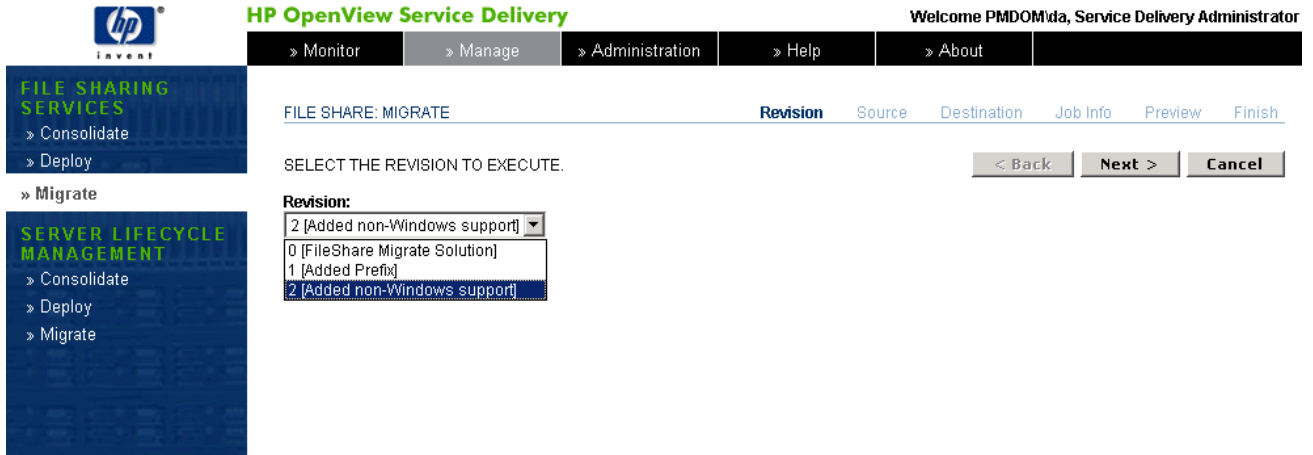
If only the EU E1 is desired for updating, then user selects the explicit method, and saves package P1. In this case P1 will EU E1 in revision 17 and EU E2 in revision 4.

For additional information, see the Solution Rollout section.

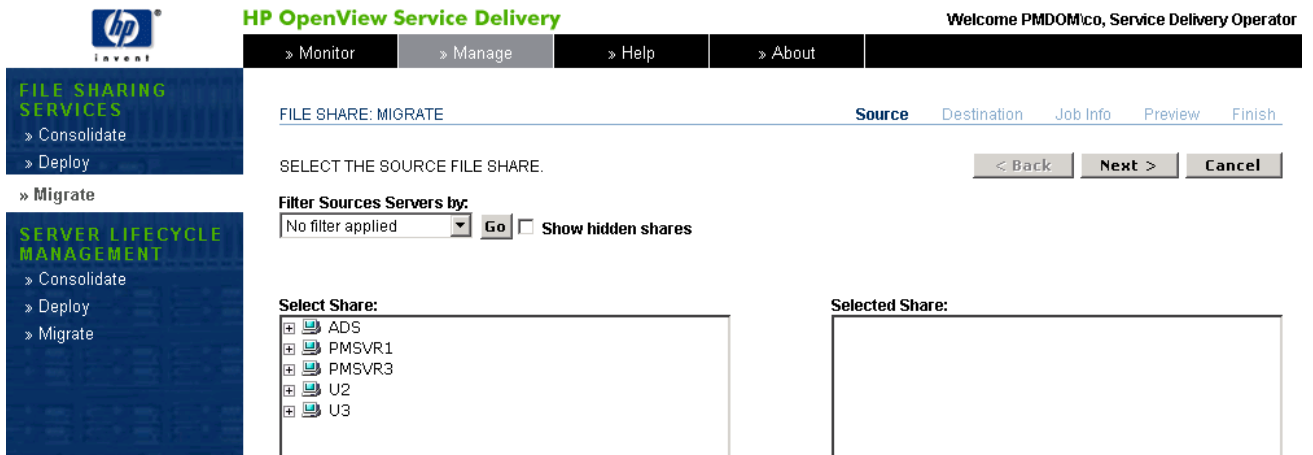
Versioning in the Service Delivery Controller Console

By design members of the ServiceDeliveryOperators group can only execute the default package (solution revision). Members of the ServiceDeliveryAdministrators and ServiceDeliveryArchitects groups can select the revision they execute.

In the following example, showing the Service Delivery Controller Console, the File Sharing Services Migrate package has a total of three revisions. Since the user running the Console is a member of the ServiceDeliveryAdministrators group, this user is able to select the revision to execute (notice that the first step in the operation is the revision selection):



However, at the same Service Delivery Controller Console, when a user member of the ServiceDeliveryOperators group is performing the same operation, only the default revision is available, and hence there is no option to select a revision. This is demonstrated in the following image (notice that the first step in the operation is not the revision selection, but actually the Source selection):



See also Package Designer versioning.

Solution Rollout

Requirements form Solution Rollout

In order to move a solution (Package) from a Service Delivery Controller server to another, both source and destination servers must be running HP OpenView Service Delivery Controller version 2.2 or later.

Introduction to Solution Rollout

In a typical solution creation process, a solution moves from the development environment to a test environment and finally to a production environment, with possible intermediate steps. The Solution Rollout feature of the HP OpenView Service Delivery supports this process.

Solution Rollout Overview

Solution Rollout is the process of enabling a solution (Package) to move from one HP OpenView Service Delivery Controller environment to another. Solution Rollout is agnostic as to whether an environment is test or production. It simply knows how to take a solution from one Service Delivery Controller environment to another.

Solution Rollout is a command line tool that has two phases. First phase is preparing and “packaging” a solution from the source Service Delivery Controller environment. Second phase is installing the new solution to a destination Service Delivery Controller environment. An implicit intermediate step is needed to transfer the created file from the source to the destination SDC environment.

Solution Rollout Details

Once the Package is working properly, has been verified and tested on the developer workbench, it is ready to use. In order to do this, the Package’s working version must be registered on the target Service Delivery Controller server.

The registration process at the target Service Delivery Controller server starts with the leaf nodes of any nested Packages and works its way upward from there. The entire Package is registered in this order, and once this has happened successfully your Package will be ready for use. The Package registration will take the latest version of each EU or nested Package of which it is comprised.


To rollout a solution from a source Service Delivery Controller environment to a destination SDC environment you need to do the following.




Solution Rollout is available only via the Service Delivery Controller CLI.

In the Service Delivery Controller CLI identify the solution (Package) version you want to rollout by issuing the following command in the CLI:

```
cao -username=<username> -password=<password> -host=<host machine>
system listpackages
```

 <username> must be a member of the ServiceDeliveryAdministrators or ServiceDeliveryArchitects groups in order to run this command. This user need not be the user logged on.

 You may cache the credentials for the username, password and host even though caching the password information is not recommended. Refer to the Service Delivery Controller documentation for details.

This produces a list of the Packages indicating the latest and default revisions from the Service Delivery Controller repository as shown in the following excerpt:

```
<?xml version="1.0"?>
<ArrayOfPackageElement xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/200
1/XMLSchema-instance">
  <PackageElement>
    <header xmlns="urn:HP-com:schema:Repository-20030423">
      <name>HPSD.Solutions.Package.ADS.ADSCapture</name>
      <latestRevision>-1</latestRevision>
      <defaultRevision>-1</defaultRevision>
    </header>
  </PackageElement>
  <PackageElement>
    <header xmlns="urn:HP-com:schema:Repository-20030423">
      <name>HPSD.Solutions.Package.ADS.ADSDeploy</name>
      <latestRevision>-1</latestRevision>
      <defaultRevision>-1</defaultRevision>
    </header>
  </PackageElement>
  ...
```

From this list identify the Package revision you want to rollout by issuing the command (in this example the File Services Migrate Package revisions are listed):

```
cao -username=<username> -password=<password> -host=<host machine>
system listpackagerevision -
name=HPSD.Solutions.Package.FileShare.Migrate
```

This creates a list of all package revisions available in the repository. For example, the above command produces the following (note that results will be different depending on the revisions a package has undergone):

```
<?xml version="1.0"?>
<PackageElement xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSc
hema-instance">
  <header xmlns="urn:HP-com:schema:Repository-20030423">
    <name>HPSD.Solutions.Package.FileShare.Migrate</name>
```

```

    <latestRevision>3</latestRevision>
    <defaultRevision>3</defaultRevision>
  </header>
  <body xmlns="urn:HP-com:schema:PackageElement-20031009">
    <revisions>
      <revision>
        <header xmlns="urn:HP-com:schema:Repository-20030423">
          <name>HPSD.Solutions.Package.FileShare.Migrate</name>
          <revision>0</revision>
        </header>
      </revision>
      <revision>
        <header xmlns="urn:HP-com:schema:Repository-20030423">
          <name>HPSD.Solutions.Package.FileShare.Migrate</name>
          <revision>1</revision>
        </header>
      </revision>
      <revision>
        <header xmlns="urn:HP-com:schema:Repository-20030423">
          <name>HPSD.Solutions.Package.FileShare.Migrate</name>
          <revision>2</revision>
        </header>
      </revision>
      <revision>
        <header xmlns="urn:HP-com:schema:Repository-20030423">
          <name>HPSD.Solutions.Package.FileShare.Migrate</name>
          <revision>3</revision>
        </header>
      </revision>
    </revisions>
  </body>
</PackageElement>

```

The above HPSD.Solutions.Package.FileShare.Migrate package has four revisions, 0-3.



By design, the default revision of a Package is its latest revision.

Finally, having selected the revision you would like to rollout, issue the following command to pack the solution (Package) and create a .cab file that contains its own self-contained installation logic.

Then copy this .cab file to the destination Service Delivery Controller server, expand it and run the `install.cmd` command. The revision selected in the packing phase will be installed on the destination.

Assume you would like to select revision #2 of the above package.

```

cao -username=<username> -password=<password> -host=<host machine>
system createpackagesetup -
name=HPSD.Solutions.Package.FileShare.Migrate -revision=2 -
outputshare=\\<destserver>\<share>

```

This command produces a “.cab” (Windows Cabinet) file and puts it in the share indicated by `\\<destserver>\<share>` as follows:

HPSD.Solutions.Package.FileShare.Migrate.2.setup.cab

- ▶ You must have sufficient write permissions on the destination share.

You must now manually copy this “.cab” file to the destination Service Delivery Controller installation. Expand it using Winzip®. Do not use the Windows Explorer because there is an issue with folder tree expansion.

- ▶ You must expand the cab file using Winzip®. Do NOT use Windows Explorer.
You can obtain Winzip® from www.winzip.com.

Expanding this file will provide the appropriate folder structure and will create `install.cmd` and `unattended_install.cmd` scripts at the root folder. Both these installation scripts will install the Package to the host Service Delivery Controller server. The installation script works by running the setup logic recursively for all components of a solution and registering this solution to the destination environment.

- ▶ You must NOT run two or more concurrent Solution Rollout installations on a given Service Delivery Controller server.

Concurrent revision installations cause unpredictable results.

- ▶ The revision (version) number of the Package on the destination server is determined by the destination server history and **not** by the source server. Thus, it can be bigger, same, or smaller, numerically, than the version on the source server.

Revision numbers are just identifying numbers used by the Service Delivery Controller system. They have no other significance.

- ▶ You must be a member of the `ServiceDeliveryAdministrators` or `ServiceDeliveryArchitects` to run this command.

Solution Rollout and Versioning

When a solution is rolled out to a Service Delivery Controller server, the version created is based on the version of the destination server, not the source. If the source Service Delivery Controller server (for example, the development server) has version 27 of a Package, whereas the production server has version 13 of the same package, then, when this package is rolled out to the production server it gets version 14 (and not 27). This Package also becomes the default package.

See Also Versioning

Additional Command Line Tools

EU Test Tool

Short Description

The EU Test Tool allows testing of individual Execution Units (EUs) in isolation, i.e., without the need to use them in an existing Package.

Description

EUs have three areas where code could run; the Execute, Cleanup, and Rollback phases. For further information on these three phases see the EU section.

The tool allows testing of the following EU lifecycle phases:

- 1 Execute
- 2 Cleanup
- 3 Rollback

Location and Command

To use the EU Test Tool, go to the SDD installation directory, ServiceDelivery\Designer and run the **EUTester.exe** command. By default the **EUTester.exe** tool is installed in:

```
C:\Program Files\HP\ServiceDelivery\Designer
```

Where Service Delivery Controller is installed in:

```
C:\Program Files\HP\ServiceDelivery
```

Security Considerations

Only members of ServiceDeliveryAdministrators or ServiceDeliveryArchitects groups can run the EU Test Tool.

Usage

EUTester.exe

Parameters

The EU Test Tool is a shell program.

The command to test an EU implementation is:

```
runeuimpl
```

The parameters required are: the path to the implementation and the type name of the class that implements the EU.

Example

```
C:\Program Files\HP\ServiceDelivery\Designer>EUTester.exe
Welcome
Commands:
? [command]
runeurep
runeuimpl
property [propertyName]
quit


>runeuimpl
Please enter EuImplPath
>>C:\temp\MyEU.dll
Please enter EuImplClass
>>MyEUNamespace.MyEUClass
```

Now you will need to enter property values, to use an available property from the properties you have defined in the EU. To do so, type the name of that property in the format: **EUClass.PropertyName**.

Resource Dictionary Tool


Short Description

The Resource Dictionary Tool allows the addition and query of resources in the Service Delivery Controller server.

 For this release of the SDD only query and enumerate functionalities are supported.

Description

The Resource Dictionary Tool provides its users with the ability to register a new resource, modify an existing resource (if permitted), query for the definition of a resource, and enumerate the resource inheritance and containment hierarchy as defined in the system.

 For this release of the SDD only query and enumerate functionalities are supported.

Location and Command

To use the Resource Dictionary Tool, go to the Service Delivery Controller installation and run the **resreg.exe** command. By default the **resreg.exe** is installed in the Service Delivery Controller installation directory, such as:

```
C:\Program Files\HP\ServiceDelivery
```

Where Service Delivery Controller is installed in:

C:\Program Files\HP\ServiceDelivery

Security Considerations

Only members of ServiceDeliveryAdministrators or ServiceDeliveryArchitects can run the Resource Dictionary Tool.

Usage

Service Delivery Controller Resource Registration Tool:

```
resreg {-a | -m ResourceId} XmlFile [DiscoveryModule]
```

-a : Register a resource definition.

-m : Modify an existing resource definition indicated by Resource ID with the new XML and Discovery Module.

Discovery Module file is not required for abstract resources.

```
resreg -q ResourceId [-v MajorVersion.MinorVersion]
[OutXmlFile DiscoveryModulePath]
```

```
resreg -eq ResourceId [-v MajorVersion.MinorVersion]
[OutXmlFile DiscoveryModulePath]
```

Query a resource by its type prefix, optionally of a given version.

Additional parameters, or using **-eq** option, imply an exact match on the resource ID and indicate the file into which the XML and the path in which the Discovery Module may be saved. Without them, all the resource IDs matching the given prefix are sent to standard output.

```
resreg -h [ResourceId]
```

Query the hierarchy information, starting at an optionally specified root.

```
resreg -c ResourceId
```

Query the containment hierarchy information, starting at a specified root.

```
resreg {-e | -r | -d [-c]} ResourceId
```

-e : Mark a resource as editable

-r : Mark a resource currently set as 'editable' back to read-only.

-d : Delete a resource definition. **-c** flag means all child resources are to be deleted as well.

```
resreg -i InputFile
```

Take all commands from the input file and process them. The commands are all in the same form as above.

```
resreg -init
```

Initialize the resource dictionary. This query fails if the resource dictionary has already been initialized.

```
resreg -?
```

Print this message.

Parameters

Please see Usage for details.

Repository Tool

Short Description

The Repository Tool allows a user to register an EU or Package, and enumerate existing versions of EUs and Packages.

Description

The Repository Tool has the ability to register a new EU, register a new version of an existing EU, a new version of an existing Package, or re-register a Package to update all of its EU and child package references, and query all the various versions of EUs and Packages.

Location and Command

To use the Repository Tool, go to the Service Delivery Controller installation and run the `RepositoryTool.exe` command.

Security Considerations

To run the Repository Tool you must be a member of `ServiceDeliveryAdministrators`, or `ServiceDeliveryArchitects`.

Usage

```
Usage: RepositoryTool --mode=TESTTYPE [OPTIONS]...
```

```
Usage: RepositoryTool [OPTIONS]
```

```
-m, --mode=TESTTYPE      The operation to use: RegisterEU,  
RegisterPackage, OutputXml, ListEUs, ListFiles, DeployFiles,  
DeployReps, DeployPackageReps, CatalogPackages, ReRegisterPackage.  
-d, --debug              Invoke debugger after initialization  
-i, --ini=STRING         Alternate INI file to use  
-o, --outputdir=STRING   Output directory for operations
```

Example

```
Register an EU  
RepositoryTool -mode=RegisterEU c:\temp\NewEU.xml  
c:\temp\NewEUImpl.dll  
List all EUs  
RepositoryTool --mode=ListEUs
```

List all Packages

```
RepositoryTool --mode=CatalogPackages
```

All other operations are for debugging and support purposes.

For an extensive example of this tool see the `EUReg.cmd` batch file in your Service Delivery Controller installation directory under the XML sub-directory.

SDD—UI Job Options Modifications

The SDD allows for modifications of job options as they are manifested in the Service Delivery Controller GUI (console) and CLI (command line). These changes are achieved by using the Package Designer Tool.

UI Job Options Overview

The Service Delivery Designer's Package Designer allows modifications of the job options. Modifications include adding or changing an external input, setting, and enabling job execution options.

The UI job options modifications functionality allows you to:

- Add a new external input
- Modify an external input
- Create new job options
- Set the default of a job option
- Set the visibility of a job option
- Reorder the job options as they appear on the job options page of the Service Delivery Controller Console

For example you may need to add another input parameter to the Service Delivery Controller File Share Migrate package. This input parameter is to be used to provide an email address for email notifications. Or you may want to set the default state of certain job options, e.g., Continue on Errors, to be different from the default as set at the product installation.

UI job options can be changed for both the GUI and the CLI of the Service Delivery Controller.

All changes affect the new revision of the package. Existing packages revisions are not affected.

When a changed package revision is moved from and Service Delivery Controller server to another using Solution Rollout, the UI modifications are moved with it.

This functionality is available for members of the `ServiceDeliveryAdministrators` and `ServiceDeliveryArchitects` groups.

Fixed vs Dynamic Input Parameters

The Package Designer allows two main types of input parameters. Fixed and Dynamic.

Typically Fixed parameters are used for package inputs that are critical to the package and should not be changed. In particular, if a package is to be called by other packages, which is the model in the Service Delivery Controller, care must be taken to select appropriately the fixed parameters. For example a package that migrates a file share from machine A to machine B will always need to have inputs that define its operands. These inputs need to be defined as fixed since without them the package can not execute correctly.

However, some inputs may be optional. For example, the above package may have email notifications options, however, if no email address is supplied it is designed to function without providing email notifications. The package can still achieve its main goal, e.g., to migrate a file share from machine A to machine B. Alternatively, certain execution options can be assumed by the package logic, e.g., the Continue on Errors option can be assumed that if is not provided it is always set (i.e., the default value is unambiguous). Therefore this parameter is set as dynamic.

The package creators must decide which parameters must be made fixed and which ones not by their knowledge of the package logic.

When a parameter is defined as Dynamic, then its display properties on the job option page in the Service Delivery Controller Console, can be managed in the Package Designer.

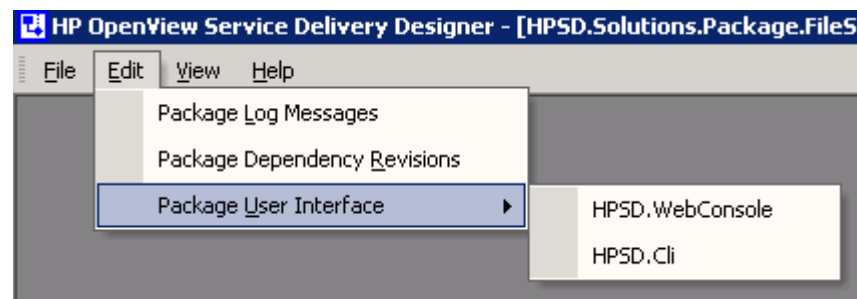
The examples in this section illustrate these cases.

GUI Job Options Modification Example—Change Default Setting

To make changes that appear in the Service Delivery Controller Console GUI, open the Package Designer and select the package to modify.

See elsewhere in this documentation on how to open packages and select the appropriate revision.

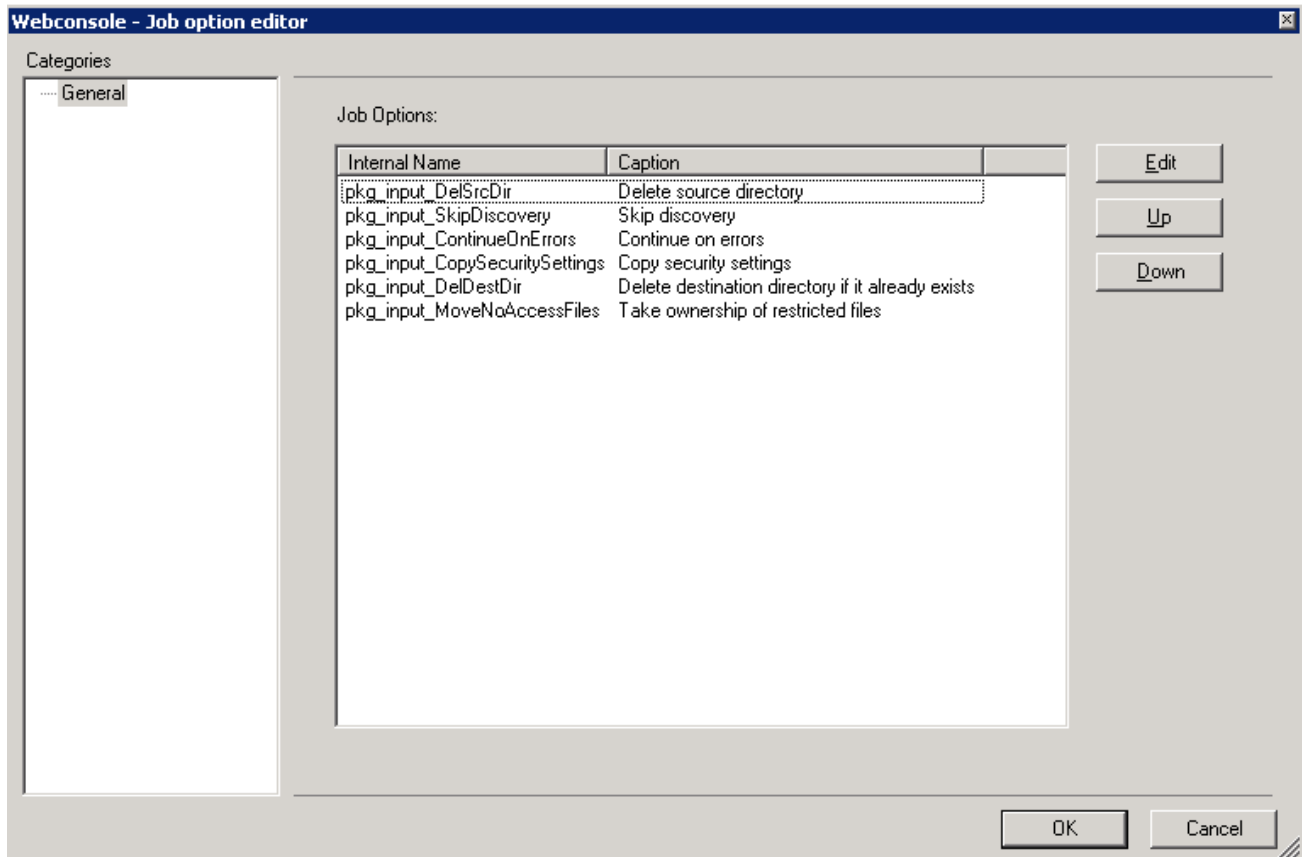
Once you have a package loaded select the command shown below:



Then select the Webconsole option.

Note that Console and WebConsole are used synonymously.

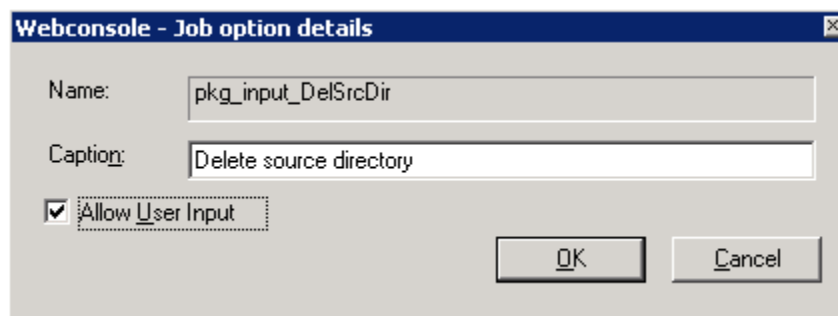
The following dialog appears:



This dialog shows all job options available for modification in this package.

Note that not all package inputs can be modified.

Select the first one, the **DelSrcDir** option and click **Edit**. Enable the **Allow User Input**:



Click **OK** and save the package into the Repository.

Now in the Service Delivery Controller Console, log in with the credentials of a member of the ServiceDeliveryAdministrators or ServiceDeliveryArchitects group and start a File Sharing Services Migrate job (based on the package just modified).

Notice the new revision that is available (and is the default one by design).

If you were to run the job using the initial Service Delivery Controller installation options the job info page would look like this:

ENTER THE JOB INFORMATION.

Job Properties

Job Name:

JobBeforeOptionsMod

Description:

Move File Share \\U2\atemp to \\U3\atemp

Job Options

- Delete source directory
- Skip discovery
- Continue on errors
- Copy security settings
- Delete destination directory if it already exists
- Take ownership of restricted files

However, in this case, select the revision just saved (on your system, the revision number will reflect the number of times a package has been saved) and create a job.

Observe the new job info page:

FILE SHARE: MIGRATE

Revision

Source

ENTER THE JOB INFORMATION.

Job Properties**Job Name:**

AfterEnablingJobOption

Description:

Move File Share \U2\atemp to \U3\atemp

Job Options

- Delete source directory
- Skip discovery
- Continue on errors
- Copy security settings
- Delete destination directory if it already exists
- Take ownership of restricted files

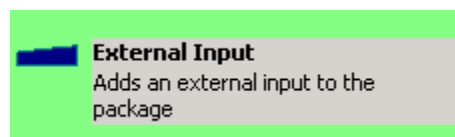
Notice that the Delete source directory is now enabled.

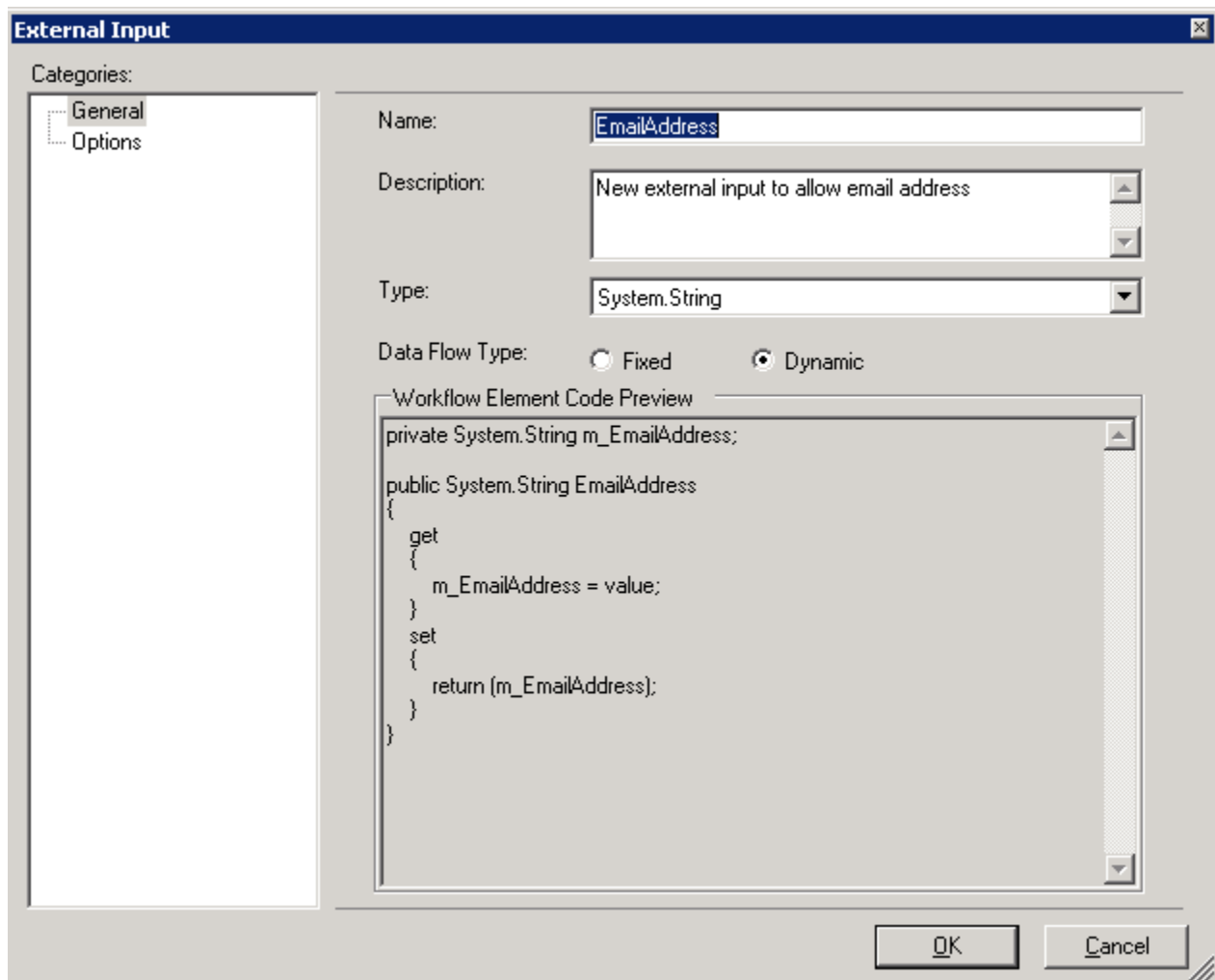
GUI Job Options Modification Example—Add New Input Parameter

To make changes that appear in the Service Delivery Controller Console GUI, open the Package Designer and select the package to modify.

See elsewhere in this documentation on how to open packages and select the appropriate revision.

First add a new input parameter in the package by adding the External Input shape. In this example a string input parameter is added to allow adding a new input for an email address. When the external input shape is dropped on the package the External Input dialog appears.





Since we are designing this package to function with or without an email address input parameter, the parameter is defined as of type Dynamic. See the section on Fixed vs Dynamic parameters elsewhere in this documentation.

Select the Options under Categories:

Here you can define various aspects of this external input for each of the three Service Delivery Controller operator types.

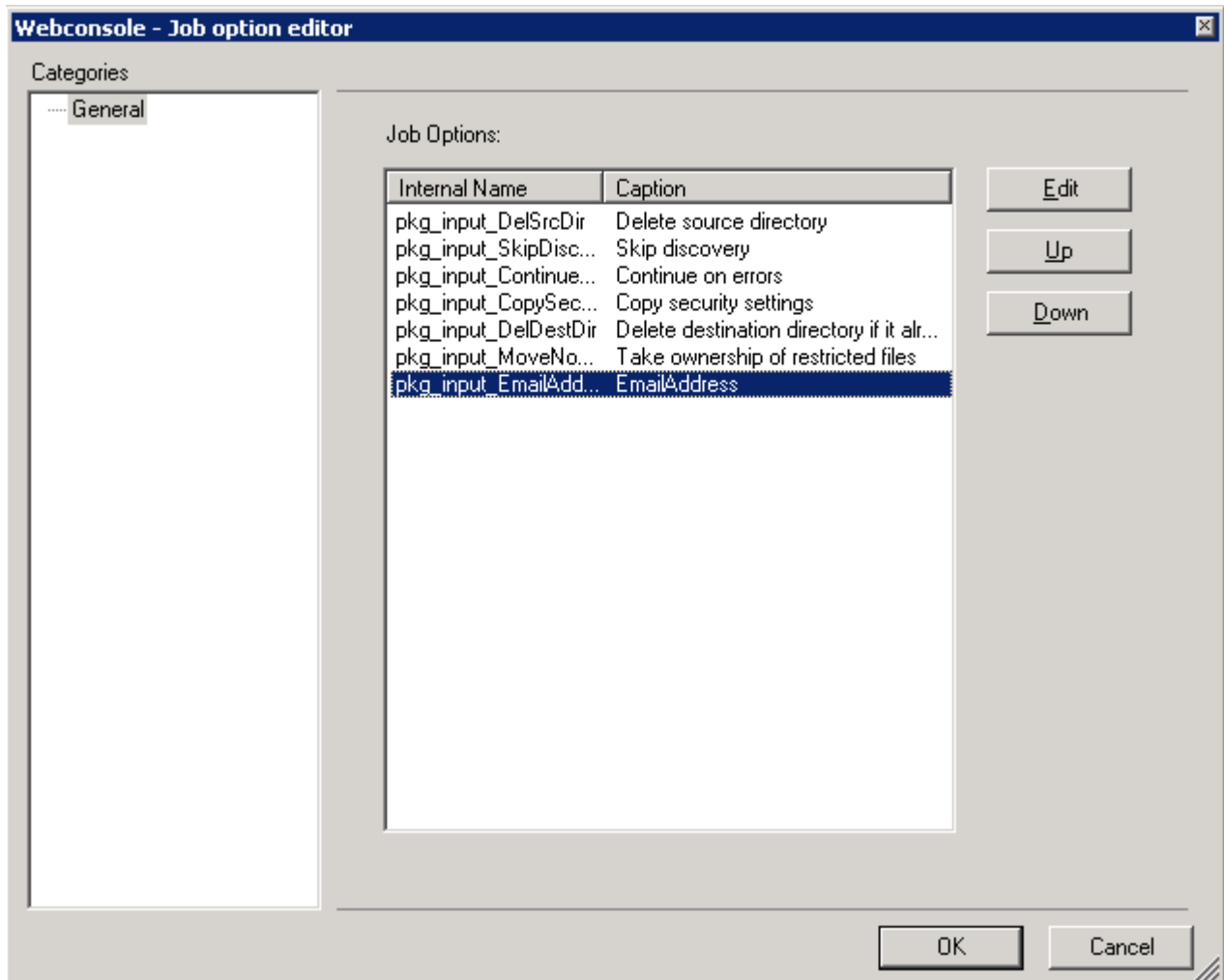
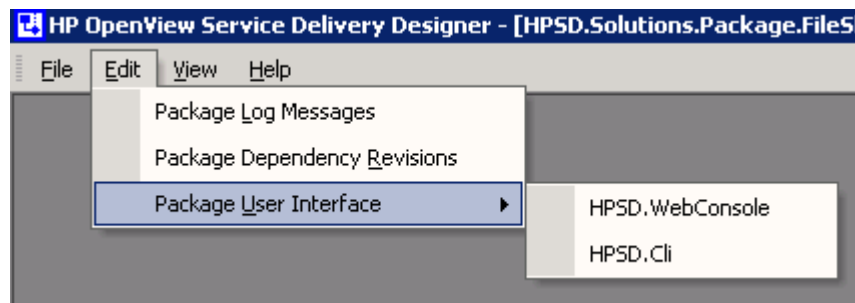
Options include:

- User Access Level
- Default Value
- Required parameter
- Length
- Validating regular expression

In this example the parameter is made as Optional and the maximum length is defined to be 50 characters. No default value is provided.

Click **OK**.

In the Package Designer open the job option editor and select the WebConsole



Notice that the newly added external input is now available as a job option.

You can change the caption and its enabled property by clicking on the Edit button.

Note that the caption is what appears on the Service Delivery Controller Console options page.

Click **OK** and save the package to the Repository.

Start the Service Delivery Controller Console, login with the credentials of a member of the ServiceDeliveryAdministrators or ServiceDeliveryArchitects and select this package revision:

FILE SHARE: MIGRATE Revision

SELECT THE REVISION TO EXECUTE.

Revision:

5 [Added External Email Address]

Note that the revision number shown on your Service Delivery Controller Console will depend on the revision history on the Service Delivery Controller you are using.

On the job info page notice that the new external input parameter with the caption defined earlier is available:

FILE SHARE: MIGRATE Revision Sol

ENTER THE JOB INFORMATION.

Job Properties

Job Name:

Description:

Job Options

- Delete source directory
- Skip discovery
- Continue on errors
- Copy security settings
- Delete destination directory if it already exists
- Take ownership of restricted files

EmailAddress

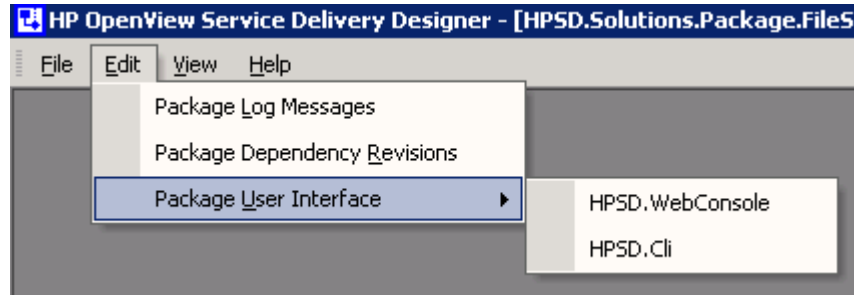
In this example the user has typed an email address of Nionio@xyz.com.

You can choose to rearrange or change the default or required behavior of this job option.

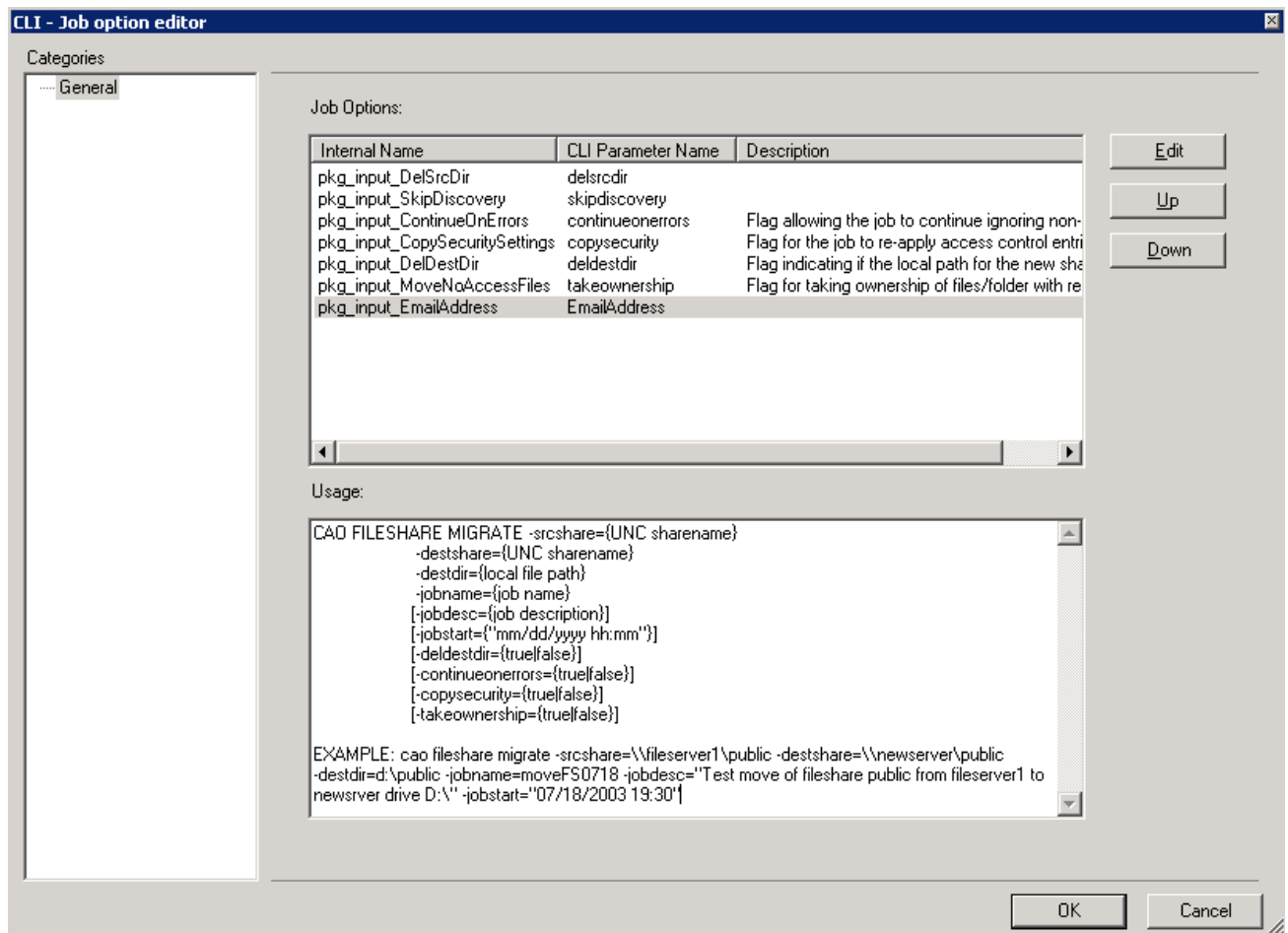
CLI Job Options Modifications

The CLI job options modification feature follows the functionality of the GUI feature and has the same design principles. In addition, it allows the definition of the command line usage and example text as shown here.

From the following Menu in the SDD Package Designer, select the CLI option.

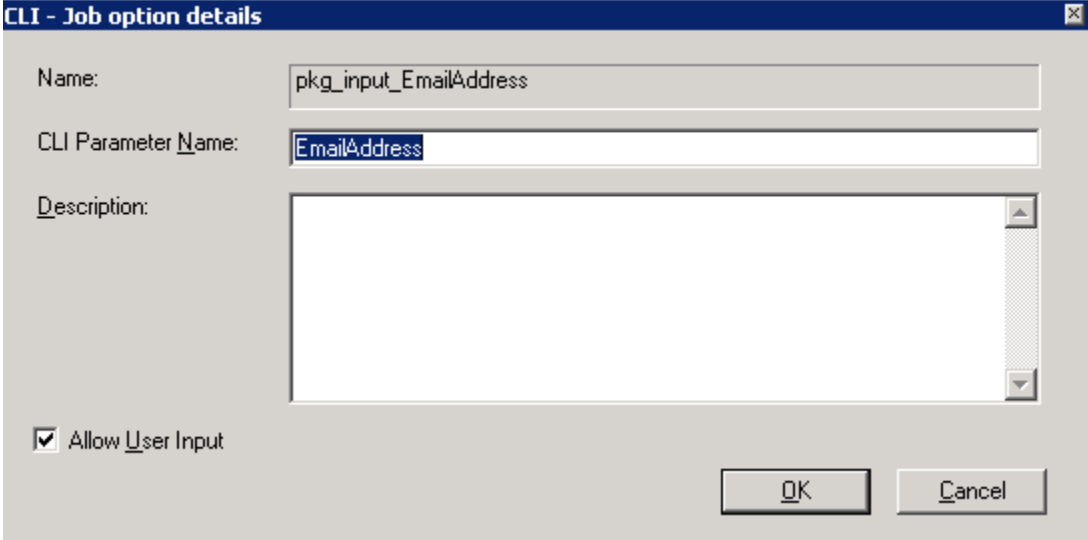


The following dialog appears:



To change the command usage and example text as it appears in the CLI, edit the Usage box in the above dialog.

To edit defaults and other props of a particular job option select the job option and click **Edit**:



The screenshot shows a dialog box titled "CLI - Job option details". It has a standard Windows-style title bar with a close button. The dialog contains the following fields and controls:

- Name:** A text box containing the text "pkg_input_EmailAddress".
- CLI Parameter Name:** A text box containing the text "EmailAddress".
- Description:** A large, empty text area with a vertical scrollbar on the right side.
- Allow User Input:** A checkbox that is checked, located at the bottom left of the dialog.
- Buttons:** "OK" and "Cancel" buttons are located at the bottom right of the dialog.

You can change the name of the CLI parameter, e.g., for localization or for CLI consistency and you can provide a description.



You must save the package to the Repository for any of these changes to take effect.

SDD Security

The SDD Security model follows that of Service Delivery Controller, in that ServiceDeliveryAdministrators and ServiceDeliveryArchitects (but not ServiceDeliveryOperators) can modify, save, and rollout packages and solutions.

Package Designer Security

Only members of the domain based groups ServiceDeliveryAdministrators and ServiceDeliveryArchitects can use the Package Designer and load and save Packages from the repository. Any user that has access to the server that the SDD is installed on may launch the Package Designer by launching Microsoft Visio 2002 and then loading the PackageDesigner.vsd.

Users other than the ServiceDeliveryAdministrators and ServiceDeliveryArchitects, get a “repository not available” error when they try to load or save to the repository.

Solution Rollout Security

Only ServiceDeliveryAdministrators and ServiceDeliveryArchitects can create a Solution Rollout (.cab) file and install a solution (on a Service Delivery Controller server).

For creating a Solution Rollout, the user may be logged on to the source Service Delivery Controller server under any valid user with the right server permissions; for example, as a ServiceDeliveryOperator, and use the necessary credentials (e.g., a user member of ServiceDeliveryAdministrators or ServiceDeliveryArchitects) when issuing the command. Note that ServiceDeliveryOperators do not have privileges to successfully issue Solution Rollout commands.

For installing a solution to a destination Service Delivery Controller server, you must log on with the credentials of a user member of ServiceDeliveryAdministrators or ServiceDeliveryArchitects.

EU Registration Security

Only ServiceDeliveryAdministrators and ServiceDeliveryArchitects can register new EUs in the Service Delivery Controller Repository.


SDD Tools Security Considerations

The SDD tools, such as EUTester, can only be used by ServiceDeliveryAdministrators or ServiceDeliveryArchitects.

SDD Execution Units

This section describes the Execution Units that are part of the library that ships with the HP OpenView Service Delivery Controller. EUs are shown categorized by area.



For a list of the available registered Execution Units in your installation, open the SDD Package Designer, drop the Use EU shape on the Package Designer and click the  button.

ADS EUs

The following EUs are in the Automated Deployment Services (ADS) name space.

ADSDeployValidator

The EU `ADSDeployValidator` validates the `ADSDeploy` package.

This EU does not include Rollback or Cleanup.

See Also ADS `ADSControllerName` EU.

CopyFile

The EU `CopyFile` copies a file from one specified remote machine to another.



This EU is specific to ADS and is separate from the `FileShare.CopyFiles` EU because the file name must be fully qualified with [\\machinename\share\filename](#).

This EU does not include Rollback or Cleanup.

See Also FileShare `CopyFiles` EU.

CopySequence

The EU **CopySequence** takes an ADS sequence string and converts it into a file, then copies that file onto an ADS server.

CopySysprepFiles

The EU **CopySysprepFiles** copies all of the Sysprep files including the INF file to the destination machine.

ExtractMachineName

The EU **ExtractMachineName** extracts the machine from a fully qualified domain name.

GenerateCaptureSequence

The EU **GenerateCaptureSequence** generates the ADS sequence for capturing the image of a device.

GenerateDeploySequence

The EU **GenerateDeploySequence** generates the ADS sequence for deploying an image to a device.

GenerateSysprepSequence

The EU **GenerateSysprepSequence** generates the Sysprep sequence for the ADS controller.

GetAdminMAC

The EU **GetAdminMAC** gets the media access control (MAC) address from an ADS device.

GetADSInstallDir

The EU **GetADSInstallDir** gets the ADS installation directory from the ADS controller.

GetBootPartition

The EU `GetBootPartition` passes in a data structure containing the disk layout, and returns the boot partition number based on the disk layout.

GetDeviceName

The EU `GetDeviceName` returns an ADS device name based on the MAC address.

GetDeviceNameByServerName

The EU `GetDeviceNameByServerName` returns the ADS device corresponding to a Windows server name.

GetDiskLayout

The EU `GetDiskLayout` returns the specified ADS device's hard drive disk layout.

InstallAgentValidator

The EU `InstallAgentValidator` installs the agent validators that will be called in the install agent Package to ensure that all the parameters are properly set.

PrepareImgVariables

The EU `PrepareImgVariables` returns a hash table containing all the image variables.

RegisterCSRImage

The EU `RegisterCSRImage` registers the selected image as a Service Delivery Controller labeled image for the ADS controller.

RegisterDevice

The EU `RegisterDevice` registers an ADS device to the ADS controller.

RunRemoteCommand

The EU `RunRemoteCommand` runs an ADS command on a remote device.

RunSequenceValidator

The EU `RunSequenceValidator` serves as the validator for the RunSequence Package.

SetDefaultTemplate

The EU `SetDefaultTemplate` sets the default template for each ADS device.

SetDeviceVariables

The EU `SetDeviceVariables` sets one or more device variable on the ADS device.

SubmitJob

The EU `submitJob` submits an ADS job to the controller.

SysprepValidator

The EU `sysprepValidator` serves as the validator for the Sysprep Package.

Domain EUs

The following EUs are in the Domain name space.

ConsolidationCheck

The EU `ConsolidationCheck` checks that the domain level of the consolidation is going to occur successfully—ensures that there is no conflict.

FileServices EUs

The following EUs are in the FileServices name space.

AddShare

The EU `AddShare` adds a share when the EU is given a server name and a share directory. It adds a specified share without permissions. This EU cannot be registered via the repository. If a user needs to change this package, they must contact HP OpenView Service Delivery for assistance.

AddTempShare

The EU `AddTempShare` adds a temporary share when the EU is given a server name and a specified path, and is an informal method of creating a share, using a GUID.

AssertFailure

The EU `AssertFailure` exits a Package and always returns false.

AssertSuccess

The EU `AssertSuccess` exits a Package and always returns true.

CopyFiles

The EU `CopyFiles` copies files given a source and destination UNC path.

DelShare

The EU `DelShare` deletes the specified share from the specified server.

DfsAdd

The EU `DfsAdd` adds the DFS link or a replica. Takes the DFS link name.

DfsAddDomainRoot

The EU `DfsAddDomainRoot` adds a domain-based DFS root to a system when the EU is given a set of parameters.

DfsAddStandaloneRoot

The EU `DfsAddStandaloneRoot` adds a stand-alone based DFS root to a system when the EU is given a set of parameters.

DfsRemove

The EU `DfsRemove` removes a DFS entry.

DfsRemoveDomainRoot

The EU `DfsRemoveDomainRoot` removes a domain-based DFS root.

DfsRemoveStandaloneRoot

The EU `DfsRemoveStandaloneRoot` removes a stand-alone DFS root.

FileShareMoveValidate

The EU `FileShareMoveValidate` takes a set of inputs, including a source server, net name, share name, and path and compares them with what the destination will be. This EU validates that no conflict exists before the move takes place.

ForceRemoveDir

The EU `ForceRemoveDir` takes the indicated directory and removes it immediately. This EU requires a server name and a path. This operation is not recoverable.

Only use this EU when Rollback is not needed.

GetNestedShares

The EU `GetNestedShares` goes to any given share and enumerates all the shares underneath it to see if there are any nested shares.

GetShareInfo

The EU `GetShareInfo` gets the share information for a specified share, including permissions.

MakeRemoteDir

The EU `MakeRemoteDir` creates a remote directory on a specified server.

NestedShareCheck

The EU `NestedShareCheck` checks for nested shares for any “root” share. Typically used before an operation commences.

RemoveDir

The EU `RemoveDir` removes a remote directory from the specified server.

SetFileAttributes

The EU `SetFileAttributes` sets the file attributes for a file or folder.

SetFileSecurity

The EU `SetFileSecurity` sets the file permissions on a file or a folder. This EU is used in the Deploy package.

SetShareInfo

The EU `SetShareInfo` sets the share information associated with a sharepoint.

WaitEU

The EU `WaitEU` adds a wait cycle.

Machine EUs

The following EUs are in the Machine name space.

AddLocalGroups

Adds the local groups that have been enumerated by `EnumLocalGroups` and adds them to the destination server.

See also `EnumLocalGroups`

ComputerNameAdd

The EU `ComputerNameAdd` adds a computer name to the server for emulation, and is used to initiate emulation.

ComputerNameDel

The EU `ComputerNameDel` removes the name that has been added to the server for emulation, and is used to terminate emulation.

EnumLocalGroups

The EU **EnumLocalGroups** enumerates the local groups on a server.

See also **AddLocalGroups**

FileSharesConsolidateCheck

This EU ensures that the shares will not conflict, and checks the share name and directory path.

FixServerName

The EU **FixServerName** ensures that the server name is being presented in the proper format. For example, it might get rid of extra \\ . It normalizes the server name if it is wrong.

GenerateNewServerName

The EU **GenerateNewServerName** generates a random number to be used as a new server name, to prevent a naming conflict. Used when a server must be renamed to fulfill the requirements of a unique name in the domain.



Even though the algorithm is usually successful at generating a previously unused server name, there is always a possibility for a server name conflict since the randomly generated server name may already exist.

Rename

The EU **Rename** renames the specified server with a new name.



This EU must not be used for NT 4.0 servers.

RenameNT4

The EU **RenameNT4** renames the specified NT4 server with a new name.



This EU is specific to NT 4.0 servers, and must not be used on Windows 2000 servers or Windows 2003 servers.

Shutdown

The EU **shutdown** shuts down the specified Windows 2000 or Windows 2003 server.

- ▶ This EU must not be used for NT 4.0 servers.

ShutdownNT4

The EU **shutdownNT4** shuts down the specified Windows NT4 server.

- ▶ This EU is specific to NT 4.0 servers, and must not be used on Windows 2000 or Windows Server 2003 servers.

ValidateUser

The EU **validateUser** validates the user credentials to ensure that they have permissions to perform operations.

Walkthrough Examples

This section provides examples and information needed to modify existing Service Delivery Controller Solutions using the Package Designer.

This section assumes you have read and familiarized yourself with the Service Delivery Controller components and understand the Service Delivery Designer's basic functionality, features, and the Package Designer Tool.

Changing a Service Delivery Controller Solution: Overview

Using the SDD software, Service Delivery Controller solutions (workflows and packages) can be changed easily. Changing a solution typically involves the following process. Note that the following is an outline. The examples below provide the details.

- i. First identify and outline the modifications needed for the solution. For example, you may want to remove or add an EU to the solution.
- ii. Next, identify the granularity of the changes. Some of these changes may be available by EUs that are built into the Package Designer. Others could be accomplished by writing custom EUs, or alternatively (and of lesser desirability), some could be achieved by writing scripts or calling out to external executables. In subsequent SDD releases additional and expanded capabilities will be offered.
- iii. If the required changes to a solution cannot be covered with the use of the above, then you must consult HP OpenView Service Delivery Support for further assistance.
- iv. The next step is to identify where in the Package (workflow) the change should be introduced.
- v. If there are requirements for new user input parameters for a solution being modified, you can incorporate them into the new solution and have them exposed in the Service Delivery Controller Console UI or CLI. See the corresponding section elsewhere in this documentation for more details.

- vi. When the new Package version is complete, you are ready to rollout the solution. Typically a Rollout involves moving the new solution from the developer environment to a testing environment and finally to a production environment. You may have more or fewer steps depending on your policies and procedures.

The sections below provide detailed examples on a spectrum of typical modifications. Note that you can also modify UI and CLI options. Please refer to the appropriate

Walkthrough Example A: Changing an Existing Solution—Removing an EU

This detailed example provides the key pieces of information needed to modify, as well as how to change an existing Service Delivery Controller Solution using the Package Designer.

In this example, we will modify an existing Service Delivery Controller solution; the File Sharing Services Migrate operation to disable the part of the solution that removes (destroys) the source directory. Thus, after the change in this example, the File Sharing Services Migrate operation will leave the source directory available (note that any DFS links will be removed). Deleting the source directory is a function of the Migrate operation that is *not* exposed in the Service Delivery Controller UI.

Identifying the Package to Be modified

Identification of the package to be modified is accomplished by using the Package Designer Tool available as part of the SDD installation. Launch the Package Designer and Load from the Repository the File Sharing Migrate solution. The solution can be accessed from the File/Load from Repository menu. In the window select the Package you want to modify; in this case File Sharing Migrate.

Identifying Necessary Changes

The second step is to identify the changes that will occur and where they should occur.

The place in the Migrate Package that destroys the source share must be identified and that step removed from the Package.

When opened in the Package Designer the File Sharing Services Migrate Package spans many pages. The first page provides a functional Package summary by displaying EUs and off-page references (scope), as well as the definition of the input parameters.

Let's first examine some of these shapes.

The first group of shapes on the diagram's top left provides the input parameters for the Package. Observe that the following, amongst others, are required input parameters: **SrcServerName**, which is the source server name, **DestServerName**, which is the destination server name, and so forth.

Next, observe the Migrate (Start) element followed by a number of shapes, EUs and/or off-page references finally terminating with the End shape. What is shown here is the complete end-to-end logic of the Migrate Package. This logic is gathered into coherent groups of the form: “First do A,” then “do B,” and so on. To draw an analogy, this is similar to the main function of a program that calls a number of subroutines to achieve its goal; you can view off-page references as analogous (but with differences) to a macro substitution in programming.

Now that the package contents are available, we need to identify the area that needs to change.

To do so, we walk through the Package’s contents.

To access off-page references simply right-click them and select GoTo. You can also go to these references by selecting the desired page.

Identifying EUs to Be Modified/Changed

When you open the File Sharing Services Migrate Package in the Package Designer you will see the solution components displayed in a flowchart on the right pane. On the left pane is a document stencil containing various shapes.

The main page (first page) details the Package flow. Going through its components towards the end you find the Cleanup sub-workflow shape, which is an off-page reference. Right-click this shape and select GoTo (or click the Cleanup page).

This takes the Package Designer to the Cleanup process. This process deletes the source server directory. This action depends on the delete source directory setting, which is an input parameter.

The next action requires you to right-click the “If” statement EU and select Delete. Now the Cleanup part of the Package is a null operation, thus achieving the desired result.

This will be true regardless of the “DelSrcDir” (delete the source directory) option setting since the code for this action has been removed.

Since this sub-workflow performs no other function, it can safely be removed from the main section of the workflow. The next step requires you to right-click at the top of the page on the “Cleanup (Start)” shape, and select Go Back. This takes you back to the main page. Select the Cleanup off-page reference and press Delete. Or right-click and select Delete. This page is now removed.

Saving the Modified Package

Having made the necessary changes, the next step is to save the Package.

Select Save to Repository from the File menu.

Provide an appropriate description and click OK. A new version of the package is saved in the Repository.

Member of the ServiceDeliveryOperators group can only run this modified revision. However, members of the ServiceDeliveryAdministrators and ServiceDeliveryArchitects groups can run this or any of the previously saved revisions.

Testing the New Package

Now that we have modified and saved the package, it is time to test it. To do so, launch Service Delivery Controller Console (you must launch the Service Delivery Controller Console that runs on the server where you modified the Package, or you must use Solution Rollout to move this version to another Service Delivery Controller installation).

Log in either as a member of the ServiceDeliveryAdministrators or ServiceDeliveryArchitects group.

Select the File Sharing Services Migrate operation. You are now presented with the screen that allows you to select the package revision. Select the revision just saved and create a job to migrate a file share from machine A (some machine in your network) to machine B (some other machine).

Upon successful completion of the job, you can observe the results as illustrated in this example.

Walkthrough Example B: Changing an Existing Solution—Adding a Prefix to a Moved Share

This detailed example provides the key pieces of information needed to change or modify existing Service Delivery Controller Solutions using the Package Designer.

In this example, we modify an existing Service Delivery Controller Solution, the File Sharing Services Migrate operation to add a prefix, of our choosing to every moved file share. In the simplistic example, the prefix is predetermined; however, with a simple change the input string could be provided at job run time by an external program such as a rules engine or a database.

Identifying the Package to Be Modified

Identification of the Package to be modified is accomplished by using the Package Designer Tool available as part of the SDD installation. Launch the Package Designer and Load from the Repository the File Sharing Migrate solution. The solution can be accessed from the File/Load from Repository menu. In the window select the Package you want to modify, File Sharing Migrate in this case. If you have already performed walkthrough example A, select revision 0, the one installed by the Service Delivery Controller installation.

Identifying Necessary Changes

The second step is to identify the changes that will occur and where they should occur.

The place in the Migrate Package that is most appropriate to provide the prefix must now be identified.

When opened in the Package Designer, the File Sharing Services Migrate Package spans many pages. The first page provides a functional summary of the Package by displaying various EUs and shapes, off-page references (scope), as well as the input parameter definitions.

Let's first examine some of these shapes.

The first group of shapes on the diagram's top left provides the input parameters for the Package. Note that the following, amongst others, are required input parameters: **SrcServerName**, which is the source server name, **DestServerName**, which is the destination server name, and so forth.

Next, we observe the Migrate (Start) element followed by a number of shapes, EUs and/or off-page references finally terminating with the End shape. What is shown here is the complete end-to-end logic, albeit at a very high level, of the Migrate Package. This logic is gathered into cohesive groups of the form: "First do A," then "do B," and so on. To draw an analogy, this is similar to the main function of a program that calls a number of subroutines to achieve its goal; you can view off-page references as analogous (but with differences) to a macro substitution in programming.

Now that the Package contents are available we need to identify the area that needs to change.

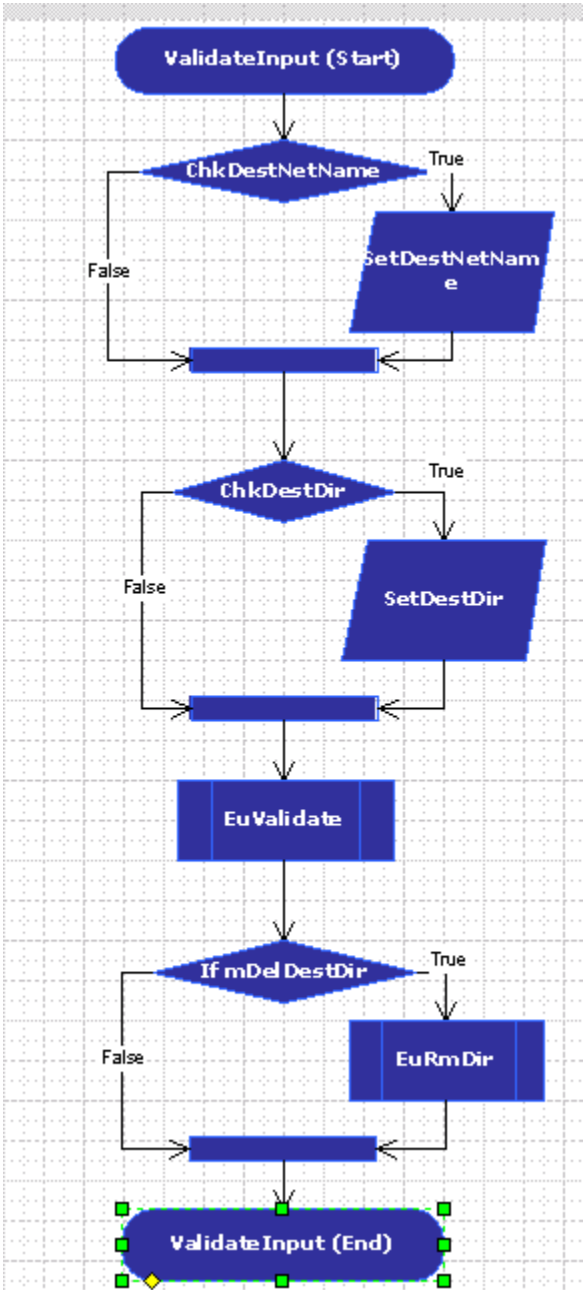
To do so, we walk through the Package's contents.

Note that to access off-page references simply right-click them and select GoTo. You can also go to these references by selecting the desired page.

Identifying the Precise Change Location

When the File Sharing Services Migrate Package is opened in the Package Designer you will see the solution's components displayed in a flowchart on the right pane. On the left pane there is a document stencil containing various shapes.

The main page (first page) details the Package flow. Going through its components you see the ValidateInput reference. Right-click it and select GoTo (or select on the ValidateInput page).



Just after the ChkDestNetName conditional ends, and before the ChkDestDir conditional starts, drop in a Variable Assignment shape.

Assign Variable
Assigns a value to a previously defined variable

When you drop the Assign Variable shape, the Variable Assignment Editor opens up as follows:

Assign Variable

Categories:

- General
- Assignment Expression

Name:

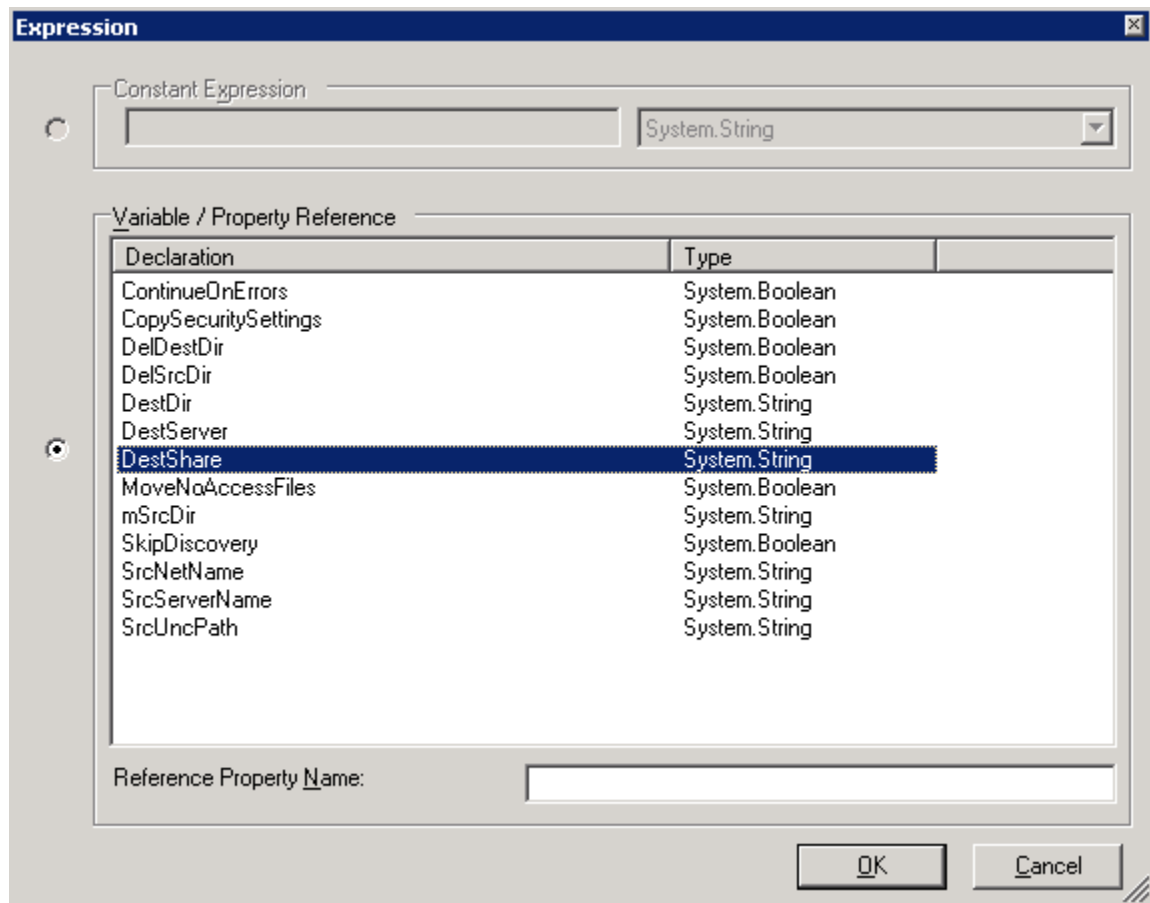
Description:

Assign To:

Type:

Workflow Element Code Preview

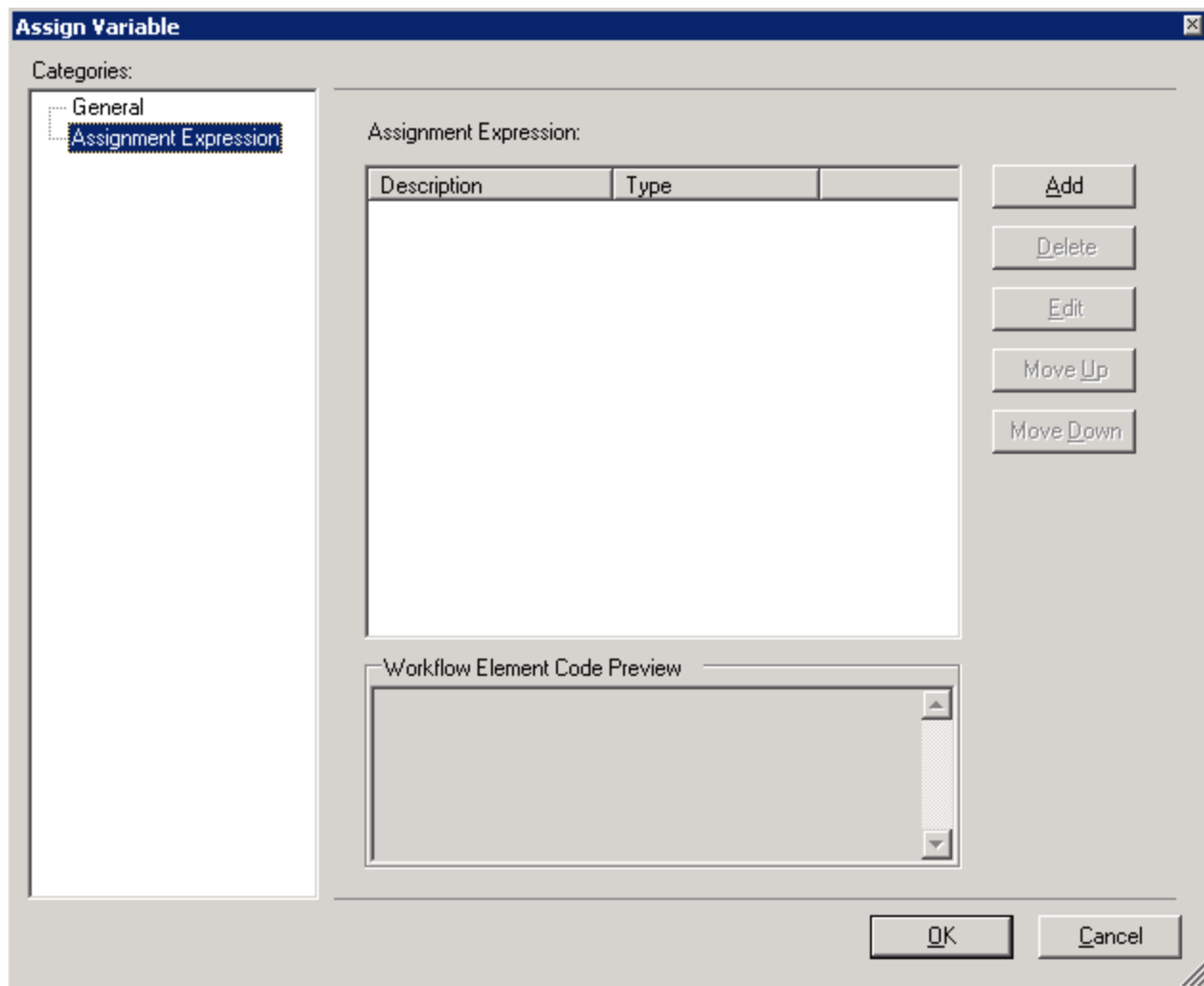
Provide a unique caption and a description. Click the Select button for the Target Declaration Name. The following expression editor appears:



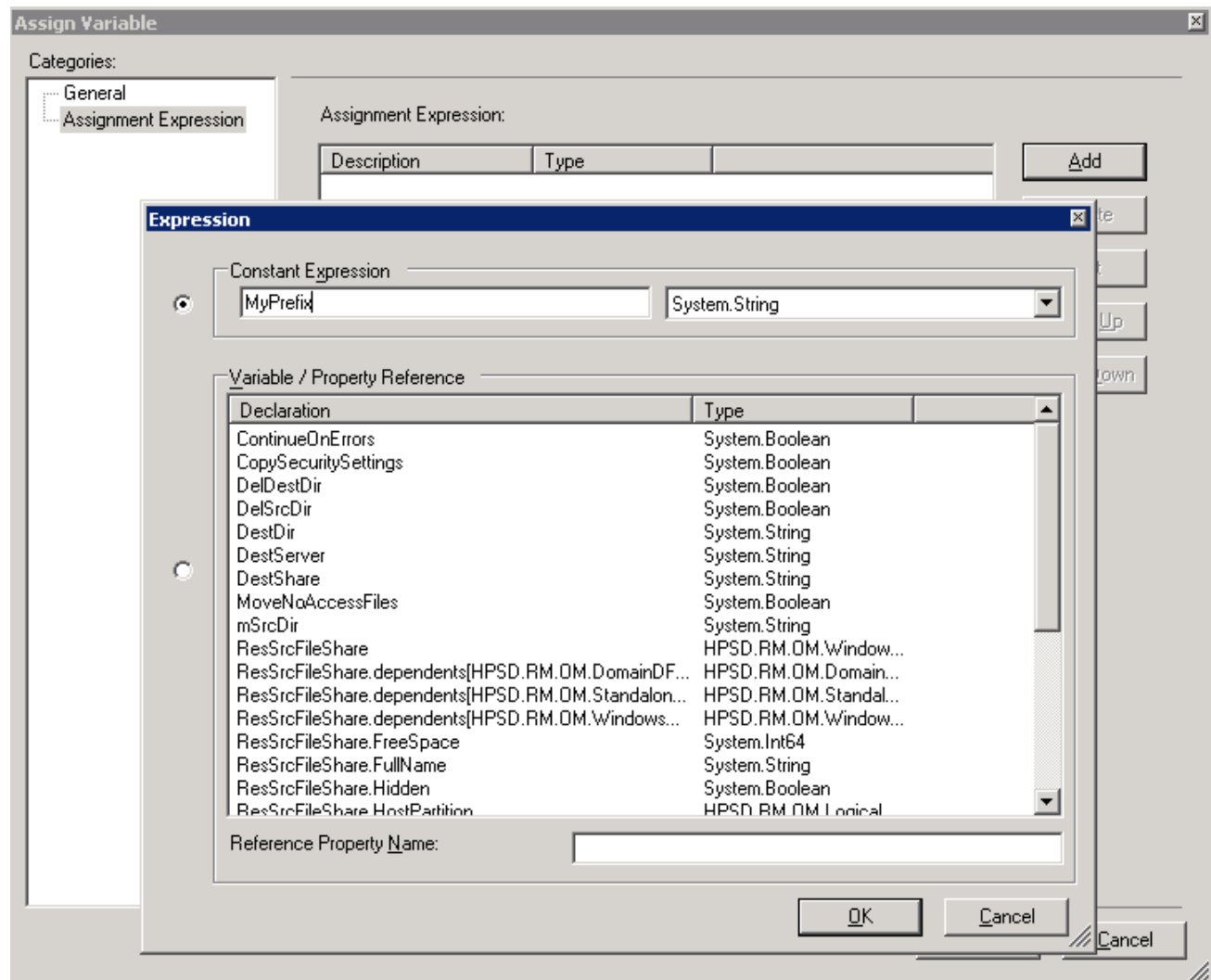
Because we want to change the Destination share name, select the DestShare as shown above.

Click **OK**.

Now click the **Assignment Expression** under Categories (top left area of the dialog). The following appears:

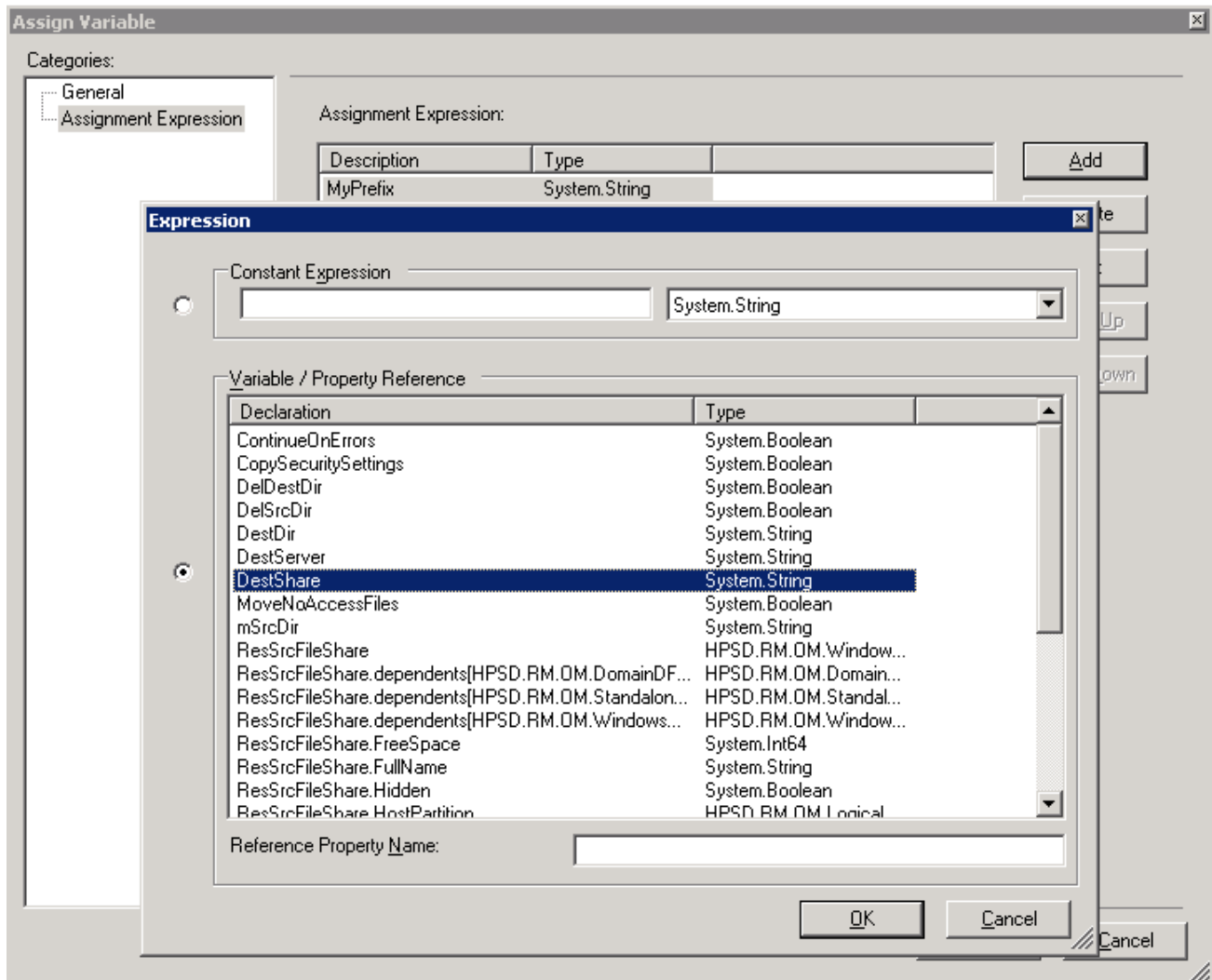


Click **Add** and type the prefix in the Constant Expression box of the Expression editor. For example, type **MyPrefix** as the prefix.

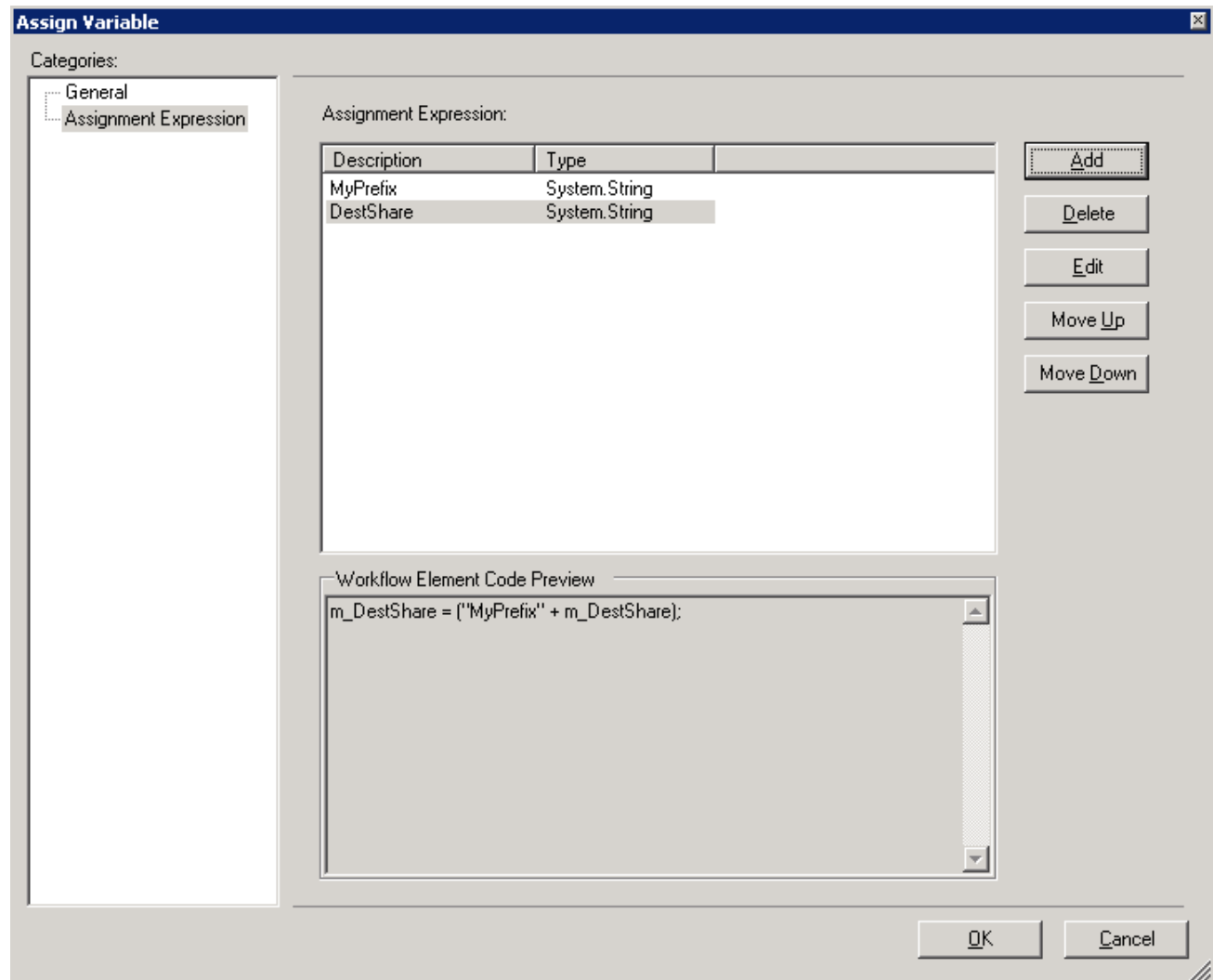


Click **OK**.

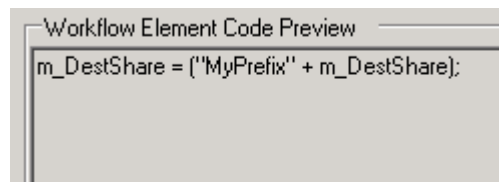
Then click **Add** (second time) and select the **DestShare** in the Expression Editor.



Click **OK**. Now the Variable Assignment editor looks like this:



Note the generated code in the statement summary box:

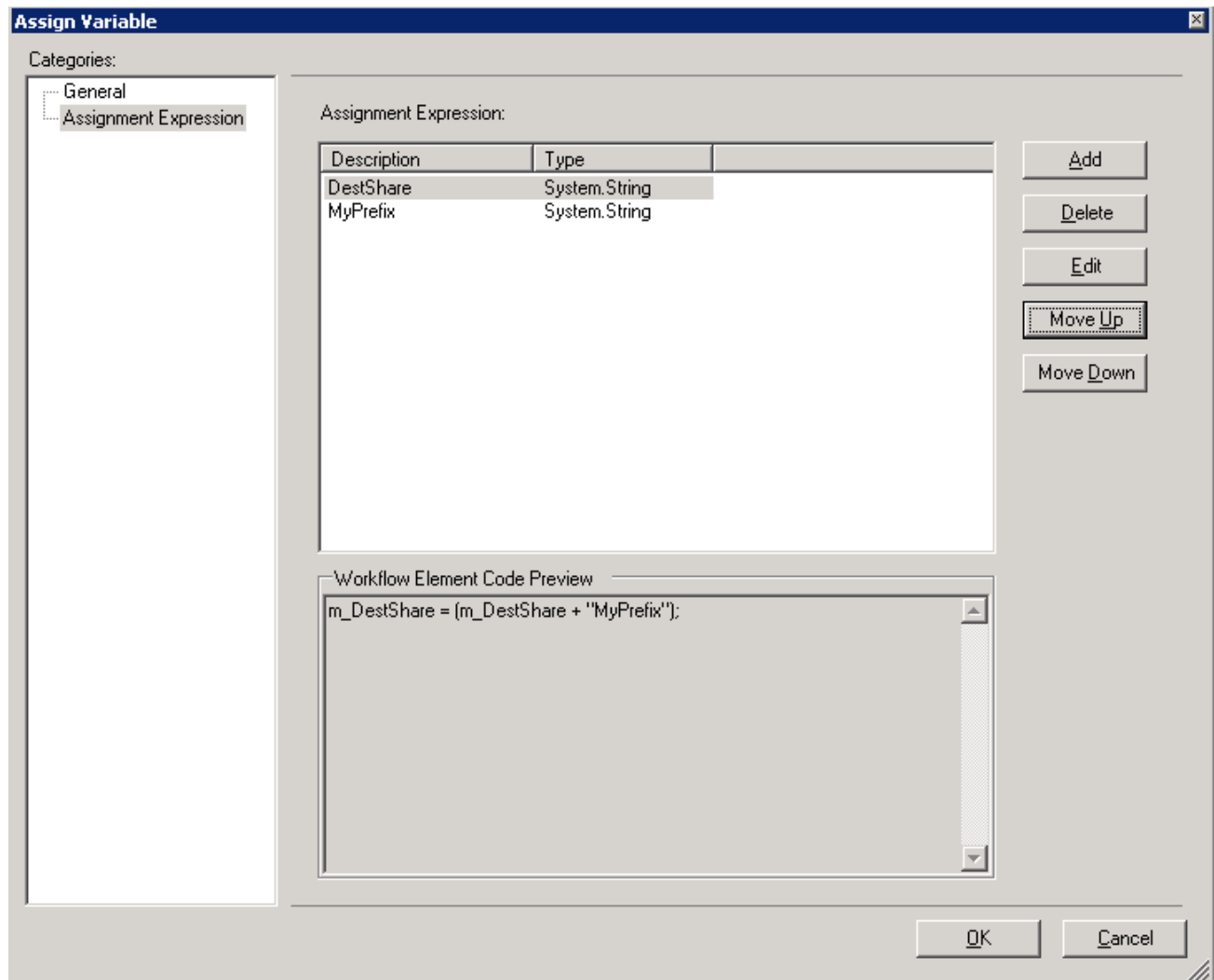


This is the actual code generated by this variable assignment.

Click **OK**.

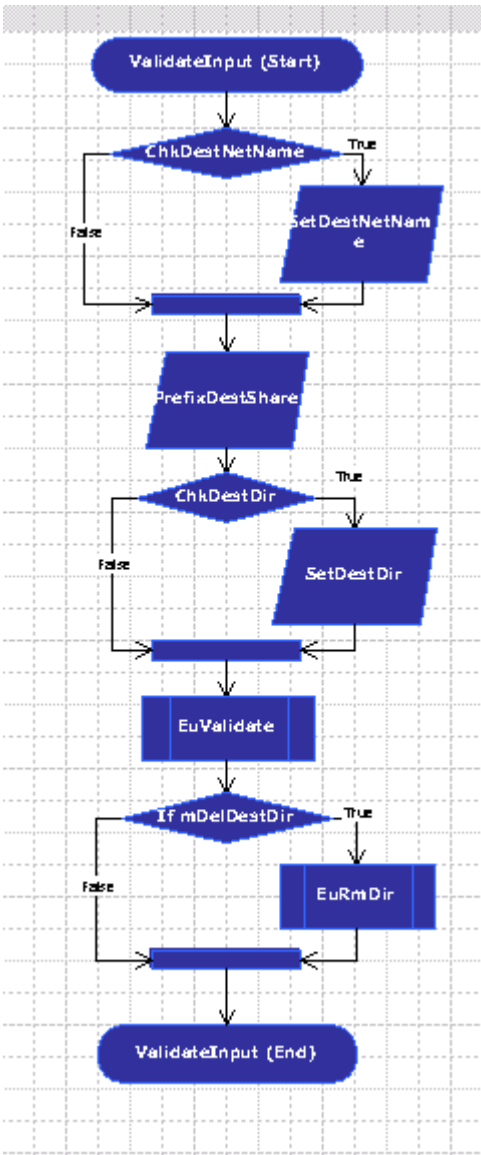


You can change this to a postfix by changing the order in the assignment expression. This can easily be achieved by selecting the DestShare and clicking on the MoveUp button as shown in the following.

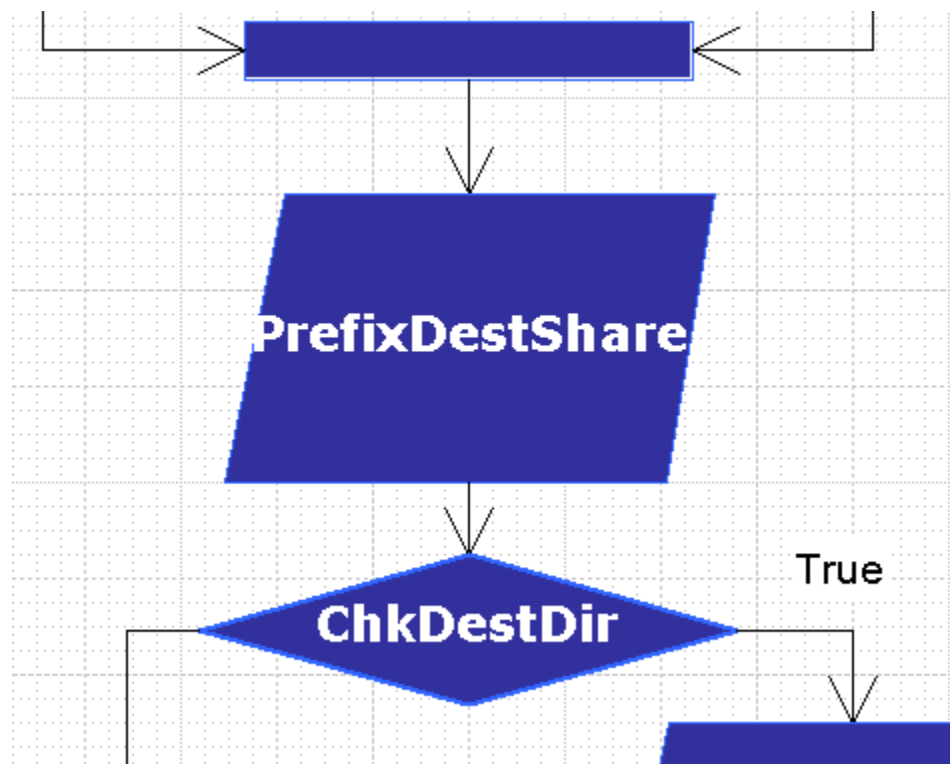


Click **Cancel** on the above screen.

The new ValidateInput sub-workflow looks like the following diagram:



Note the shape that has been added, the PrefixDestShare as shown below:



At this point the changes can be saved to the repository.

Saving the Modified Package

Having made the necessary changes the next step is to save the Package.

Select Save to Repository from the File menu.

Provide an appropriate description and click OK. A new Package version is saved to the Repository.

Testing the New Package

Now that the Package has been modified and saved, it is time to test it. To do so, launch the Service Delivery Controller Console (the Service Delivery Controller Console must be launched on the same server used to modify the Package, or Solution Rollout must be used to move this version to another Service Delivery Controller installation).

Log in either as a member of the ServiceDeliveryAdministrators or ServiceDeliveryArchitects group.

Select the File Sharing Services Migrate operation. You are now presented with the screen that allows you to select the package revision. Select the revision just saved and create a job to migrate a file share from machine A (some machine in your network) to machine B (some other machine).

Upon successful completion of the job, you can observe that the destination share has the defined prefix. Now repeat the above steps, but when selecting the Package revision to run, select revision 0, the original Package version. Run another migration. Observe that the share is removed verbatim.



Appendix A – SampleEU Code and Markup

This appendix contains the SampleEU code and markup as used in examples in this tutorial.

SampleEU Code

Listing A-1: SampleEU Code Listing

```
using System;
using System.IO;
using Microsoft.Win32;
using HPSD.PA.Developer;
using HPSD.Core.P1;

namespace HPSD.Solutions.EU.Samples.Impl{
    /// <summary>
    ///
    /// </summary>
    /// <remarks>
    ///
    /// </remarks>
    public class SampleEU : IEU
    {
        private string    m_strInput = "Hello World!";
        private string    m_strFileName = "";
        private string    m_strDirectory = "";
        private string    m_strMachineName= "";
        private string    m_strTempFileName = "";
        private bool     m_bWasCreated = false;

        /// <summary>
        /// Input string to be appended
        /// </summary>
        public string Input
        {
            get
            {
                return (m_strInput);
            }
            set
            {
                m_strInput = value;
            }
        }
    }
}
```

```
    }  
}  
  
/// <summary>  
/// The machine name where the file resides  
/// </summary>  
public string MachineName  
{  
    get  
    {  
        return (m_strMachineName);  
    }  
    set  
    {  
        m_strMachineName = value;  
    }  
}  
  
/// <summary>  
/// Wheher the file was created or not  
/// </summary>  
public bool WasCreated  
{  
    get  
    {  
        return (m_bWasCreated);  
    }  
    set  
    {  
        m_bWasCreated = value;  
    }  
}  
  
/// <summary>  
/// The file where the string will be appended  
/// </summary>  
public string FileName  
{  
    get  
    {  
        return (m_strFileName);  
    }  
    set  
    {  
        m_strFileName = value;  
    }  
}  
  
/// <summary>  
/// Directory name where the file is in  
/// </summary>  
public string Directory  
{  
    get  
    {  
        return (m_strDirectory);  
    }  
    set  
    {  
        m_strDirectory = value;  
    }  
}
```

```

    /// <summary>
    /// Temporary file name
    /// </summary>
    public string TempFileName

    {
        get
        {
            return (m_strTempFileName);
        }
        set
        {
            m_strTempFileName = value;
        }
    }

    /// <summary>
    /// Copies the original file to a temporary file
    /// Appends the string to the original file
    /// </summary>
    /// <remarks>
    ///
    /// </remarks>
    /// <returns>
    ///
    /// </returns>
    public bool Execute()
    {
        string strFile = FullFileName();

        m_strTempFileName = Path.GetTempFileName();

        try
        {
            StreamWriter w = null;
            if (!File.Exists(strFile))
            {
                // Create a file to write to.
                w = File.CreateText(strFile);

                if (w != null)
                {
                    WasCreated = true;
                }
            }
            else
            {
                File.Copy (strFile, m_strTempFileName,

                true);

                w = File.AppendText(strFile);
            }
            if (w == null)
            {
                return false;
            }

            w.Write (Input);
            w.Close ();

        }
        catch
        {
            return false;
        }
    }

```

```

        return true;
    }

    /// <summary>
    /// Deletes the temporary file created in the Execute method
if exists
    /// </summary>
    /// <remarks>
    ///
    /// </remarks>
    /// <returns>
    ///
    /// </returns>
    public bool Cleanup()
    {
        try
        {
            if (WasCreated)
            {
                File.Delete (m_strTempFileName);
            }
        }
        catch
        {
            return false;
        }

        return true;
    }

    /// <summary>
    /// Copies the temporary file to the original file
    /// </summary>
    /// <remarks>
    ///
    /// </remarks>
    /// <returns>
    ///
    /// </returns>
    public bool Rollback()
    {
        string strFile = FullFileName ();
        try
        {
            if (WasCreated)
            {
                File.Delete (strFile);
            }
            else
            {
                File.Copy (m_strTempFileName, strFile,
true);
                File.Delete (m_strTempFileName);
            }
        }
        catch
        {
            return false;
        }

        return true;
    }

```



```

    /// <summary>
    /// Builds the full path of the file name
    /// </summary>
    /// <returns></returns>
    private string FullFileName ()
    {
        string strFile = null;

        if (MachineName.Length == 0)
        {
            strFile = Path.Combine (Directory, FileName);
        }
        else
        {
            string tmp = Directory;
            tmp.Replace (':', '$');
            tmp = string.Format ("\\{0}\\{1}",
MachineName, tmp);
            strFile = Path.Combine (tmp, FileName);
        }

        return strFile;
    }
}
}
}

```

SampleEU Markup

Listing A-2: SampleEU Markup Listing

```

<eu:execution-unit version="1.0" xmlns:eu="urn:HP-
com:schema:ExecutionUnit-20030429">
  <eu:name>SampleEU</eu:name>
  <eu:namespace>HPSD.Solutions.EU.Samples</eu:namespace>

  <eu:implementation>HPSD.Solutions.EU.Samples.Impl.SampleEU</eu:implem
entation>
    <r:abstract xmlns:r="urn:HP-com:schema:Repository-20030423">
      <r:category>Samples</r:category>
      <r:version major="2" minor="1" />

      <r:description>
        Implements a sample execution unit, appends the input
string to the specified file.
      </r:description>

      <r:authors>
        <r:author name="HP" email="Support@mycompany.com"
company="My Company" />
      </r:authors>
    </r:abstract>
    <id:interface xmlns:id="urn:HP-com:schema:InterfaceDescription-
20030429">
      <id:loader location="%HPSD_BINS_DIR%\Loader\EULoader.exe" />
      <id:variables>
      <id:input>
        <id:variable id="Input" class="System.String">
          <id:description>
            The string to be appended into the output file.
          </id:description>
        </id:variable>
      </id:input>
    </id:interface>
  </eu:implementation>
</eu:execution-unit>

```

```

        </id:description>
        <id:example>
            Any text such as "Hello world!"
        </id:example>
    </id:variable>
    <id:variable id="FileName" class="System.String">
        <id:description>
            The name of the file to where the string will be
            appended.
        </id:description>
        <id:example>
            foo.txt
        </id:example>
    </id:variable>
    <id:variable id="Directory" class="System.String">
        <id:description>
            The directory where the file is.
        </id:description>
        <id:example>
            foo.txt
        </id:example>
    </id:variable>
    <id:variable id="TempFileName" class="System.String">
        <id:description>
            The path of the temporary file. This is not an
            input variable and used
            by rollback and cleanup.
        </id:description>
        <id:example>
            c:\temp.fileName
        </id:example>
    </id:variable>
    </id:input>
    </id:variables>
    </id:interface>
</eu:execution-unit>

```



Appendix B – AddShare EU Code and Markup

This appendix contains the code and mark up of the EU AddShare as used in examples in this tutorial.

AddShare Code

Listing B-1: AddShare Code Listing

```
using System;
using System.IO;
using Microsoft.Win32;
using HPSD.PA.Developer;
using HPSD.Core.P1;
namespace HPSD.Solutions.EU.Samples.Impl
{
    public class AddShare : IEU
    {
        private string mServerName = null;
        public string ServerName
        {
            get
            {
                return mServerName;
            }
            set
            {
                mServerName = value;

                if (mServerName != null)
                {
                    if (!mServerName.StartsWith("\\\\"))
                    {
                        mServerName = "\\\\" + mServerName;
                    }
                }
            }
        }
    }
}
```

```

    }
    }
}
private NetShare.ShareInfo2 mShareInfo = null;
public NetShare.ShareInfo2 ShareInfo
{
    get
    {
        return mShareInfo;
    }
    set
    {
        mShareInfo = value;
    }
}
private ExecState mExecStatus = ExecState.Begin;
public ExecState ExecStatus
{
    get
    {
        return mExecStatus;
    }
    set
    {
        mExecStatus = value;
    }
}
public enum ExecState : int
{
    /// <summary>
    ///     Begin Execution
    /// </summary>
    Begin = 0,

    /// <summary>
    ///     Share Already Exists
    /// </summary>
    ShareExists = 1,

    /// <summary>
    ///     Share Added
    /// </summary>
    ShareAdded = 2,

    /// <summary>
    ///     End Execution
    /// </summary>
    End,
}
public bool Execute()
{
    // Enter
    mEuLoader.Logging.SubmitFunctionEnter();

    // Assume failure
    bool retval = false;

    // Set Execution Status
    SetExecStatus(ExecState.Begin);

    // Validate the share to be added.
    if (!NetShare.Exists(mServerName, mShareInfo.NetName))
    {
        // Add the Share;
    }
}

```

```

if (NetShare.Add(mServerName, mShareInfo))
{
    // Share Added
    SetExecStatus(ExecState.ShareAdded);

    // Log the Message
    mEuLoader.Logging.SubmitOperatorLog(
        LogMessage.FileServicesAddShareSuccess,
        mShareInfo.NetName, mServerName);

    retval = true;
}
else
{
    // Log the Error
    mEuLoader.Logging.SubmitOperatorLog(
        LogMessage.FileServicesAddShareFailed,
        mShareInfo.NetName, mServerName);
}
}
else
{
    // Share Exists;
    SetExecStatus(ExecState.ShareExists);

    // Log the Error
    mEuLoader.Logging.SubmitOperatorLog(
        LogMessage.FileServicesAddShareExists,
        mShareInfo.NetName, mServerName);
}

// Leave & Return
mEuLoader.Logging.SubmitFunctionLeave(retval);
return retval;
}
public bool Rollback()
{
    //Enter
    mEuLoader.Logging.SubmitFunctionEnter();

    // Assume success
    bool retval = true;

    // If the Share has been added then remove it
    if (mExecStatus == ExecState.ShareAdded)
    {
        // Delete the Share;
        if (NetShare.Delete(mServerName,
mShareInfo.NetName))
        {
            // Log that it was successful
            mEuLoader.Logging.SubmitOperatorLog(
                LogMessage.FileServicesAddShareRemoved,
                mShareInfo.NetName, mServerName);
        }
        else
        {
            // Log the error
            mEuLoader.Logging.SubmitOperatorLog(
                LogMessage.FileServicesAddShareRemoveFailed,
                mShareInfo.NetName, mServerName);
        }

        retval = false;
    }
}

```

```

    }
    // Leave & Return
    mEuLoader.Logging.SubmitFunctionLeave(retval);
    return retval;
}
public bool Cleanup()
{
    return true;
}
private void SetExecStatus(
    ExecState Status)
{
    mExecStatus = Status;
    mEuLoader.SetRollback();
}
}
}
}

```

AddShare Markup

Listing B-2: AddShare Markup Listing

```

- <eu:execution-unit version="1.0" xmlns:eu="urn:HP-com:schema:
ExecutionUnit-20030429">
Name of the EU
  <eu:name>AddShare</eu:name>
Namespace of the EU
  <eu:namespace>HPSD.Solutions.EU.FileServices</eu:namespace>
Implementation of the EU
  <eu:implementation>HPSD.Solutions.EU.FileServices.Impl.AddShare</eu:
:implementation>
- <r:abstract xmlns:r="urn:HP-com:schema:Repository-20030423">
Category that this EU is a part of
  <r:category>FileServices</r:category>
Version information. This version is provided by the creator of the
EU.
  <r:version major="2" minor="2" />
Description
  <r:description>Adds a sharepoint</r:description>
Author Information
- <r:authors>
  <r:author name="AuthorName" email="Author@Mycompany.com"
company="MyCompany" />
  </r:authors>
</r:abstract>
- <id:interface xmlns:id="urn:HP-com:schema:InterfaceDescription-
20030429">
  <id:loader location="%HPSD_BINS_DIR%\Loader\EUloader.exe"/>
- <id:variables>
EU Inputs. These will be shown to the Package Designer, when this EU
is used. They must match the parameters used by the EU.
- <id:input>
- <id:variable id="ServerName" class="System.String">
  <id:description>The name of the server to remove the share
from.</id:description>
  <id:example>For a given UNC path: "\\TestSrv3\public", the
ServerName would be "TestSrv3"</id:example>
  </id:variable>

```

```
- <id:variable id="ShareInfo
class="HPSD.Win32.Netapi32.NetShare.Share Info2">
  <id:description>The ShareInfo object describing the share we are
creating.</id:description>
  </id:variable>
</id:input>
EU State Information
- <id:state>
- <id:variable id="ExecStatus"
class="HPSD.Solutions.EU.FileServices. Impl.AddShare.ExecState">
  <id:description>Execution Status</id:description>
  </id:variable>
</id:state>
</id:variables>
</id:interface>
</eu:execution-unit>
```